MASTER'S THESIS

# Master of Science in Advanced Mathematics and Mathematical Engineering

**Title:** Architectural Layout Design with Spectral Methods

**Author:** Júlia Folguera Profitós

**Advisor:** Dr. Jordi Cortadella Fortuny

**Department:** Computer Science

**Academic year:** 2019-2020



UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Facultat de Matemàtiques i Estadística

Universitat Politècnica de Catalunya
Facultat de Matemàtiques i Estadística

Master in Advanced Mathematics and Mathematical Engineering

Master's thesis

# Architectural Layout Design with Spectral Methods

**Júlia Folguera Profitós**

Supervised by Dr. Jordi Cortadella Fortuny

September, 2020

I would like to express my gratitude to Jordi Cortadella for his project proposal —which I have deeply enjoyed— his constant support and his help during the process. I would also like to thank and acknowledge my peers, my family and friends who have been cheering me up and giving me encouragement, not only in the making of this paper, but throughout my studies. A special thank you to Anaïs, Sergi and Laia.

## Abstract

The design of a floor plan is an important phase of the design of an apartment, although it is very complex due to the quantity of variables involved. The mathematical version of this problem is the architectural layout problem which is an optimization problem that aims to find the best layout of a one-level apartment given an objective function. Since this problem is non-convex and high-dimensional, in this thesis, we will try to find a good initial input so that another algorithm can find a minimum as close as possible to the global minimum. To do so, first we will find the best positions of the centers of the rooms given the design and proximity relations between rooms set by the user using spectral graph drawing. Then, we will preprocess the result so that it can be used as an input of another algorithm.

### Keywords

Floor plan, architectural layout, spectral methods, graph visualization, non-convex optimization

# Contents


**1 Introduction** **3**

**2 Problem statement and graph model** **5**

**3 Resolution of the problem** **7**

3.1 Initial guess: Rooms' centers . . . . . 8

3.1.1 Constraints . . . . . 13

3.2 Definition of the rooms' walls . . . . . 22

3.2.1 Finding the bounding box of minimum size . . . . . 22

3.2.2 Minimizing the intersection . . . . . 26

**4 Example: A medium sized apartment** **30**

**5 Conclusion and prespective** **34**

5.1 Future work . . . . . 34

5.2 Acquired knowledge . . . . . 34

**A Proofs of propositions** **36**

**B Program of the example of chapter 4** **39**

**C Program to produce the no-intersection model of chapter 4** **44**

**D Program to produce the minimum boundary box model of chapter 4** **47**

# 1. Introduction

One of the most important aspects when designing an apartment is the design of its layout. It is a complex process, due to the quantity of variables that are involved in it, as well as the fact that not only technical objectives — such as maximizing the natural light in the living areas — are taken into account, but also aesthetic ones.

In order to automate it, the problem has been mathematically modeled. It is called *space layout planning*, or floor planning and it is an optimization problem which objective is to find the best arrangement of a set of elements (the rooms) taking into account a given criteria. For instance, we could want to have an apartment with maximum natural light in the bedroom and kitchen, with the bathroom near the kitchen and with given areas for each of the rooms. To mathematically solve the problem, an objective function is put as the weighting of this parameters, and the problem is thus constructed as an optimization problem.

This problem has been widely studied, giving rise to different approaches that will be later summarised. However, since the dimension of the variables is usually large and the problem non-convex, no good algorithm is known.

The idea of this project came from paper [11], in which, given a graph topology, a gradient-descent algorithm is used to find a good minimum of a function, that summarizes and weights both the aesthetic and technical objectives; and an evolutionary algorithm is used to find the topology with the smallest minimum.

However, in this project we take a different approach. Given an initial input (figure 1) that will establish some design preferences of proximity between rooms, we will construct a new graph with the best location of the centers of the rooms (figure 2). Then, we will proceed to find the best dimensions of the rooms (figure 3). Our objective is not to find a feasible floor plan, but to find a good initial input, given the designer objectives, so that another algorithm can find the local minimum to be as close as possible to the global optimum, producing for instance, figure 4.

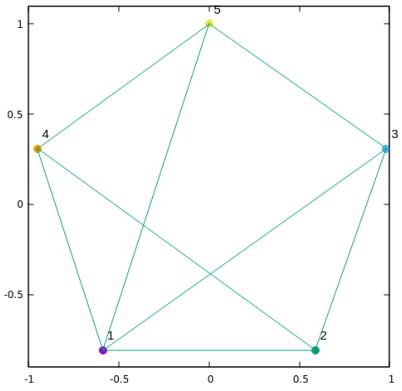


Figure 1: Initial topology graph

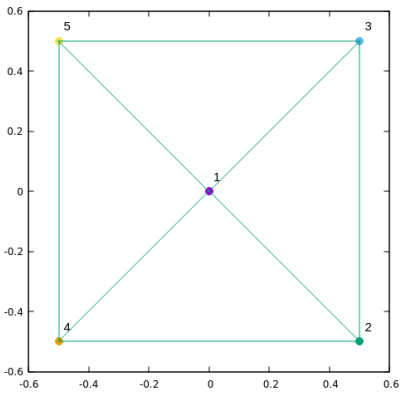


Figure 2: Best location of the centers of the rooms

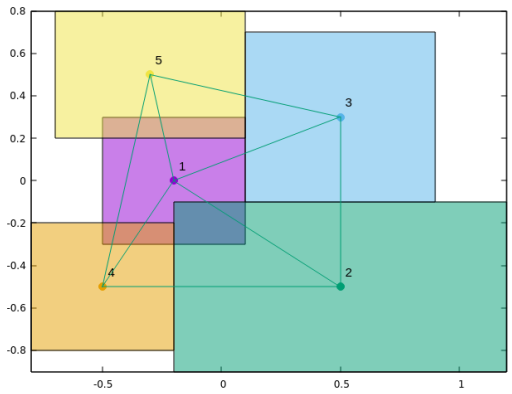


Figure 3: Dimensions of rooms and preprocess


Figure 4: Final floor plan

In this project, we will start by giving further details of the problem and constructing the mathematical model we will work with. We will then proceed to the resolution of the problem.

We will find the best location of the centers of the rooms using a physical analogy. With this we will not only try to maintain as much as possible the proximity and design properties the user has input, but also will have a aesthetic purpose.

Secondly, we will define the dimensions of each room and preprocess them to ease the work that the gradient-based algorithm would have to do. Two ways of doing this will be presented. Finally, some examples and the corresponding results will be shown.

# 2. Problem statement and graph model

To simplify the model we will only consider rectangular floors and rooms. This restriction is not a hard limitation, since if an L-shaped room is needed, two rectangular rooms can be used instead. This model is usually divided in the topological and geometrical part.

The topology part deals with which rooms should be contiguous. It can be expressed in a graph where adjacencies represent the importance of being nearer; whereas nodes indicate the centers of the rooms. In some approaches, such as in the paper [11], it is usual to optimize the topology in order to find which graph gives the best optimum of the objective function. However, in this project the topology part will be set by the user in order to satisfy their design preferences. For instance, we could want the following conditions:

- The dining room should be an important room with connections with other rooms.
- The dining room should be near the entrance and also near the kitchen.
- A general bathroom should be near the public spaces.
- Two bedrooms should have a private shared bathroom.

For these conditions, the resulting graph is:

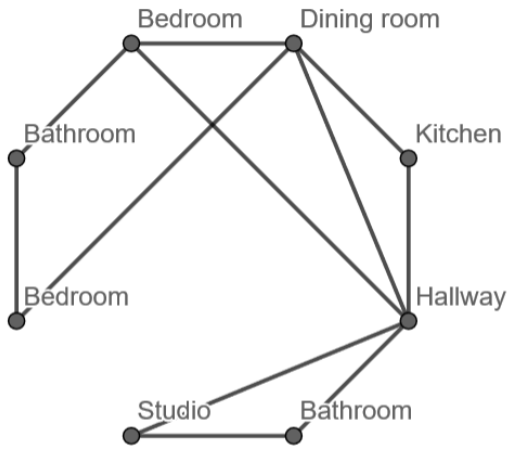


Figure 5: Topology of the house layout

Edges of the graph could also have weights to explain the degree of importance given to the nearness of their rooms. The usual weight is 1 and would represent a high degree of importance, but we can also use weights in $[0, 1]$.

On the other hand, the geometric part tries to find the optimum location and dimensions of the rooms. In order to define a room we will only need its center, $(x, y)$ and its width and length.

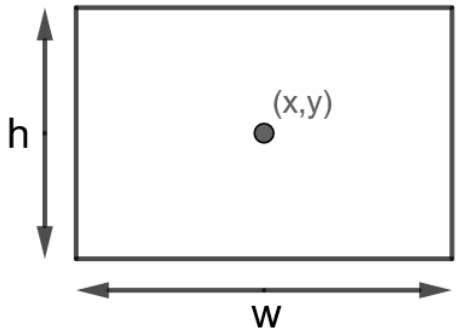


Figure 6: Model of a room

Although there are different type of rooms and each of them has different uses and properties, we will not distinguish them at this level, but later on. For instance, if we set a room to be a corridor, we will treat it as any other room and then add a constraint in its width or length so that it is not too wide.

In order to achieve the design objectives we need to define constraints, some of which are topological and others geometric. The idea of some of them has been found in [2]. For each constraint, we begin explaining what it is, why and when is it useful and, on chapter 3.1.1, how can we implement them. These are:

**Fixed points.**

Fixed points are used so as to impose that the points can not move. In this project they are used in order to orient the rooms. This can be reached by adding the four cardinal points and relating the rooms that we want to be oriented to the corresponding points. For instance, if we want the studio to be oriented to the south, so that it has more natural light, we can connect its vertex to the north vertex.

**Vertical/horitzontal aligment.**

To impose the centers of some rooms to be aligned, it can be useful when you already know that some similar rooms have to be in the same side of a corridor.

**Clustering.**

Clusters are used to group some rooms together. The clustering can have different forces depending on the result you want to get. An example of this is when there is a hallway dividing the apartment, or when trying to find the best layout of a pair of apartments.

**Fixed subgraph**

Fixed subgraphs can be useful when it is clear the distributions of a part of the apartment. The distribution of the fixed subgraph can not be changed, although you can apply to the subgraph translations, rotations and homotheties.

**Constraint to a given convex region.**

In some cases it can be useful to impose a point to be always inside a given convex region. This is the case, for instance, when we want to limit the distance between a pair of rooms.

# 3. Resolution of the problem

There have been made several attempts to solve the problem since more than 50 years ago, according to [9]. Each one taking a different approach. The first important attempt was done in 1999 at the paper [1] by rethinking the problem as a physical one. This approach will be better explained in chapter 3.1, as it is similar to what we want to do in this project.

Another type of strategy is called discursive grammar, developed at the paper [4]. It consists in defining a grammar and a semantic. The grammar is a set of substitutions rules that produces a final statement, whereas the semantic defines which words (that is, which designs) are correct. An heuristic algorithm chooses, at each step, amongst all the possible grammar rules in order to find the one that will produce the best output.

One could also divide the space in a grid and assign each of the squares a purpose. This is a discret, combinatorial problem and there are different types of algorithms to solve it, such as stochastic algorithms.

Finally, another usual approach is the constrained-based. It consists of defining some constraints and limitations and using, for instance, a gradient-based algorithm for the geometric part; and an evolutionary algorithm to find the topology of the graph that produces the best geometric optimum. This is the option used in the paper that gave rise to this project.

In our project we will take a different approach for the topological and geometric part. As we have already said, the topology part will be set by the user to satisfy their design needs. On the other hand, the geometry section will be solved using spectral methods, nonlinear optimization methods and a combination of the two. It will have two parts:

Firstly, we will find the best location of each node of the graph, that is, the optimum location of the centers of the rooms. This will be done taking a physical point of view of the graph: each of the vertices could be a body electrically charged that feels attraction and repulsion to the others. The attraction is given between adjacent vertices, since edges can be seen as springs; whereas the repulsion needed so that vertices do not collapse is caused by its electric charge. Then, a so called force algorithm will be used to find the equilibrium state of the force graph. Afterwards, we will also work with the prior introduced constraints. To do so, we will use spectral methods, sometimes combined with nonlinear optimization. We will see that in some cases we do not achieve the global optimum, but something near it.

Secondly, we need to find the best dimensions for each room and retouch its location to have a better minimum. The final solution will not be a feasible floor plan, but a good input for another algorithm. The solution, if possible, will be a global minimum. To find it, we will model the problem as a nonlinear optimization problem and then use a solver, Couenne.

Couenne is a solver for mixed-integer nonlinear programming. It attempts to find the global optimum of a constrained, non-convex optimization problem using a spatial branch and bound method. This reformulates the problem so that it is easier to obtain a lower bound on the optimal solution. Then the problem is linearized so that we can apply a branch and bound algorithm. For the simple problems that we will

need to solve, Couenne finds the optimum global solution in not too much time, whereas for some more complex problems, we will have to stop before it finds it.

To use Couenne, or any other nonlinear solver, we will need an interface, a modelling language. In this project we have used AMPL. This translate our optimization problems into something that the solvers can read, whereas our input is human readable. The models introduced in it will be produced by a program, since there are too many constraints and variables.

## 3.1 Initial guess: Rooms' centers

In this chapter we will explain a method to obtain the best node position of the graph. As explained above, we will reinterpret our graph, that has the centers of the rooms as nodes and whose edges indicate the level of importance of the closeness between the rooms. We will start by explaining the optimization problem to be solved and then the algorithm that gives the solution. The proofs of the propositions can be found in appendix A.

Force directed algorithm for drawing graphs have been widely studied, as explained for instance in the book [7], although the model and algorithm that will be presented is based on the paper [8] that uses spectral theory and the eigenvectors of matrices associated to the graph in order to draw graphs. The use of this physical parallelism can help understand and visualize the process.

Consider the graph $G = (V, E)$ with vertices $|V| = n$ and weighted edges $\omega \in [0, 1]$ that is summarized in the adjacency matrix $A$. We need to find $x_1, x_2 \in \mathbb{R}^n$ so that $(x_1(i), x_2(i))$ is the best location of the vertices $i \in V$.

Given the Euclidean distance between vertices $i$ and $j$ defined as

$$d_{ij} = \sqrt{(x_1(i) - x_1(j))^2 + (x_2(i) - x_2(j))^2}$$

the problem that we need to solve is:

$$\begin{aligned} \underset{x_1, x_2 \in \mathbb{R}^n}{\text{minimize}} \quad & E(x_1, x_2) = \frac{\sum\limits_{(i,j)\in E} \omega_{ij} d_{ij}^2}{\sum\limits_{i<j} d_{ij}^2} \\ \text{such that:} \quad & \mathrm{Var}(x_1) = \mathrm{Var}(x_2) \\ & \mathrm{Cov}(x_1, x_2) = 0 \end{aligned} \tag{1}$$

The numerator of the objective function corresponds to the attraction created by the springs, whereas the denominator produces the repulsion that guarantees that the vertices are far away enough as if they were electrically charged. The first constraints are the equivalence between the variances where

$$\mathrm{Var}(x) = \frac{1}{n} \sum_{i=1}^{n} \left( x(i) - \frac{1}{n} \sum_{j=1}^{n} x(j) \right)^2$$

This is done so that the vertices are equally scattered along both of the edges. The second constraint helps give more information to each of the axes, as it implies that there is no correlation between the axes and it requires the covariance to be zero,

$$\mathrm{Cov}(x_i, x_j) = \frac{1}{n}\sum_{i=1}^{n} x_1(i)x_2(i) - \frac{1}{n^2}\sum_{i=1}^{n}\sum_{j=1}^{n} x_1(i)x_2(j)$$

To eliminate a degree of freedom, we need an additional requirement:

$$\sum_{i=1}^{n} x_p(i) = 0 \qquad p = 1, 2$$

Which can be rewritten as $x_p^T x_p = 0$. This can help reformulate the other constraints. Since

$$\begin{aligned}\mathrm{Var}(x) &= \frac{1}{n}\left(\sum_{i=1}^{n} x(i)^2 + \frac{1}{n^2}\sum_{j=1}^{n} x(j)^2 - \frac{2}{n}\sum_{j=1}^{n} x(i)x(j)\right) = \frac{1}{n}\sum_{i=1}^{n} x(i)^2 + \frac{1}{n^2}\sum_{i=1}^{n}\sum_{j=1}^{n} x(i)x(j) \\ &= \frac{1}{n}\sum_{i=1}^{n} x(i)^2 = \frac{1}{n}x^T x\end{aligned}$$

Then the first constraint is equivalent to $x_1^T x_1 = x_2^T x_2 = 1$ as the last equality would only mean a resizing. The second constraint can also be simplified:

$$\mathrm{Cov}(x_1, x_2) = \frac{1}{n}\sum_{i=1}^{n} x_1(i)x_2(i) - \frac{1}{n^2}\sum_{i=1}^{n}\sum_{j=1}^{n} x_1(i)x_2(j) = \frac{1}{n}\sum_{i=1}^{n} x_1(i)x_2(i)$$

So it means $x_1^T x_2 = 0$. To continue we need to introduce the Laplacian. First we consider $D$ a diagonal matrix with the vertices' degree in the diagonal and recall that $A = \{\omega\}_{ij}$. Then, the Laplacian of the graph is

$$L = D - A$$

Then, since

$$\begin{aligned}x^T Lx = x^T(D-A)^T x = x^T Dx - x^T Ax &= \sum_{i=1}^{n} deg_i x(i)^2 - \sum_{i=1}^{n}\sum_{j:(i,j)\in E} w_{ij}x(i)x(j) \\ = 2\sum_{(i,j)\in E}\omega_{ij}x(i)^2 - 2\sum_{(i,j)\in E}\omega_{ij}x(i)x(j) &= \sum_{(i,j)\in E}\omega_{ij}\left(x(i)-x(j)\right)^2\end{aligned}$$

We have

$$x_1^T Lx_1 + x_2^T Lx_2 = \sum_{(i,j)\in E} w_{ij}d_{ij}^2$$

This implies that the numerator of the objective function is $x_1^T L x_1 + x_2^T L x_2$. To simplify the denominator,

$$\sum_{i<j}(x(i)-x(j))^2 = \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n}\left(x(i)^2+x(j)^2-2x(i)x(j)\right) = \frac{1}{2}\left(2n\sum_{i=1}^{n}x(i)^2 - 2\sum_{i=1}^{n}\sum_{j=1}^{n}x(i)x(j)\right)$$
$$= n\sum_{i=1}^{n}x(i)^2 = nx^Tx$$

Finally, the model explained at (1) can be expressed:

$$\begin{aligned} \underset{x_1,x_2\in\mathbb{R}^n}{\text{minimize}} \quad & \frac{x_1^T L x_1 + x_2^T L x_2}{x_1^T x_1 + x_2^T x_2} \\ \text{such that:} \quad & x_1^T x_1 = x_2^T x_2 = 1 \\ & x_1^T x_2 = x_1^T 1_n = x_2^T 1_n = 0 \end{aligned}$$

We only need one further change before solving the optimization problem, which is to use D-normalize vectors. As the numerator will tend to centralize higher degree vertices, the degree normalized denominator will help scatter them. The optimization problem is:

$$\begin{aligned} \underset{x_1,x_2\in\mathbb{R}^n}{\text{minimize}} \quad & \frac{x_1^T L x_1 + x_2^T L x_2}{x_1^T D x_1 + x_2^T D x_2} \\ \text{such that:} \quad & x_1^T D x_1 = x_2^T D x_2 = 1 \\ & x_1^T D x_2 = x_1^T D 1_n = x_2^T D 1_n = 0 \end{aligned} \tag{2}$$

We already have the optimization problem to be solved. The solution has to do with the eigenvectors of the Laplacian, and thus, we call this process spectral graph drawing. We need a prior result before finding the solution of the optmization problem.

**Lemma 3.1.** *Consider $v_1, \dots, v_{p-1} \in \mathbb{R}^n$ and $X \in \mathbb{R}^{n\times p}$ a matrix whose columns are orthogonal. Then there exists a matrix $Y$ whose columns are also orthogonal and holds:*

1. *For all $2 \le k \le p$, the column $k$ of $Y$, $Y^k$ is orthogonal to $v_1, \dots, v_{k-1}$.*
2. *For all $n \times n$ matrix $A$, $tr\,(X^TAX) = tr\,(Y^TAY)$.*

This lemma can be used to prove that the solution of the optimization problem is the two smallest generalized eigenvectors of (L,D), that is the $x$ such that there exist $\lambda$ and $x^TL = \lambda Dx$.

Recall that we will order the eigenvectors according to their respective eigenvalues. Then if $\lambda_1 \le \cdots \le \lambda_n$ we will say that the smallest eigenvector is $v_1$.

A generalized version of our problem is:

**Theorem 3.2.** *Consider a symmetric positive definite $n \times n$ matrix $A$ and a diagonal positive definite matrix $B \in \mathbb{R}^{n \times n}$ and $p < n$. The solution of the optimization problem*

$$\begin{aligned} \underset{x_1,\ldots,x_p \in \mathbb{R}^n}{\textit{minimize}} \quad & \frac{\sum_{i=1}^{n} x_i^T A x_i}{\sum_{i=1}^{n} x_i^T B x_i} \\ \textit{such that:} \quad & x_1^T B x_1 = \cdots = x_p^T B x_p \\ & x_i^T B x_j = 0 \quad \forall 1 \leq i, j \leq p \end{aligned}$$

*is the smallest generalized eigenvectors of (A,B).*

Now that we know why it is called spectral graph drawing, it would be useful to gather more knowledge about the Laplacian. Some of its properties are:

1. L is real symmetric, then it has $n$ eigenvalues and all of them are real and its eigenvectors are orthogonal.
2. The columns and files of the Laplacian sum 0, then $L1_n = 0$, thus $1_n$ is an eigenvector of $L$ with eigenvalue 0.
3. The multiplicity of the eigenvalue 0 is the number of connected components and hence in our graphs will be one.
4. L is positive semidefinite so its eigenvalues are nonnegative.
5. The generalized eigenvalues of (L,D) are in [-1,1]. The generalized eigenvalues of (L,D) are the eigenvalues of $D^{-1}L$. Then the result is due to the Gershgorin theorem, which states that the eigenvalues have to be in the discs of center the elements of the diagonal and radius the sum of the files without the diagonals.

Finally another property that is held is:

**Proposition 3.3.** *The generalized eigenvectors of $(L, D)$ are also the generalized eigenvectors of $(A, D)$ with reverse order.*

Now we will derive a force algorithm. Deriving $x^T L x = \sum_{(i,j)\in E} \omega_{ij} \left(x(i) - x(j)\right)^2$,

$$\frac{\partial x^T L x}{\partial x(i)} = 2 \sum_{j:(i,j)\in E} \omega_{ij}(x(i) - x(j))$$

So the minimum is in

$$x(i) = \frac{\sum_{j:(i,j)\in E} \omega_{ij} x(j)}{deg_i}$$

That means, the best location for $i$ is at the centroid of its neighbors. The first algorithm seems to be created by Tutte and consists on move, at each iteration, all the vertices to the centroid of its neighbors. In

order to avoid degenerative solutions, a convex face of the drawing should be fixed. The algorithm presented on [8] does not need this special vertices, but takes into consideration that degenerate solutions should be avoided, that is, eigenvector $1_n$ of degenerate eigenvalue 0. We avoid this one, imposing orthogonality against it.

As we have seen, the solution to the problem is the generalized eigenvectors of (L,D), that is, $x$ so that $Lx = \lambda Dx$, or $\lambda x = D^{-1}Lx$, that can be rewritten point to point as

$$\lambda x(i) = x(i) - \frac{\sum_{j:(i,j)\in E} \omega_{i,j} x(j)}{deg_i}$$

So we have to move the vertexes that are in the center of the drawing, to the centroid of their neighbors; and the ones that are on the boundary of the drawing have to be shifted a little bit outside. As we have seen in proposition 3.3, we can swap $L$ for $A$. But now we do not search for the smallest eigenvector, but for the biggest one, since the order is reversed. Then, the algorithm that helps finding the solution to the first axes, should be:

$x_0$ such that $x_0^T D 1_n = 0_{n\times n}$
$x_{i+1} = D^{-1}Ax_i$

The process converges to the dominant eigenvector, that is, the one with the greatest eigenvalue, using the absolute value. But this could be either $v_2$ or $v_n$ and we want to converge to $v_2$. Since the eigenvalues are in $[-1, 1]$ we can find the eigenvectors of

$$\frac{1}{2}\left(Id + D^{-1}A\right)$$

In this way, the eigenvectors will be the same but the eigenvalues will be in $[0, 1]$ and so the dominant will have to be $v_2$. In fact, the dominant would be the eigenvalue 0, but as it is a degenerate one, we will omit it. On the other hand, it is not needed to impose orthogonality in each iteration, but because of the numerical stability, it is better to do so. The process described above is used for unidimensional drawing, but although we want a bidimensional drawing, the procedure is the same for the second axes.

Instead of stopping the algorithm when the distance between points is too small, it is better to stop it when the change of direction is negligible.

Finally the algorithm would be:

```
/* N = number of vertices, P = dimension of the drawing +1, in our case 3
   u = 3xN matrix with the eigenvectors. So we already have u[0]=1_N.
   u1, u2 = 2xN matrix with our computations*/
/* To compute the top eigenvectors of (1/2)D^-1 A*/

u2[0], u2[1]=/*Random inicialization*/

for(iter=1;iter<1000 && (prod[0]<tol || prod[1]<tol);iter++){

    for(i=0;i<(P-1);i++)
        u1[i] = u2[i]

    /*D-orthogonalize against previous veps*/
    for(i=0;i<(P-1);i++){
        for(j=0;j<=i;j++)
            u1[i]=u1[i]-u1[j]^T Du[j] / u[j]^T Du[j]
    }

    /*Multiply with 0.5(I+D-1A)*/
    for(i=0;i<(P-1);i++)
        u2[i] = 1/2 (Id + D^-1 A) u1[i]

    /*Normalize*/
    for(i=0;i<(P-1);i++)
        u2[i] = u2[i] / modul(u2[i])

    /*Compute the dot product*/
    for(i=0;i<(P-1);i++)
        prod[i] = u1[i]*u2[i]

    /*Save the results*/
    for(i=0;i<(P-1);i++)
        u[i+1] = u1[i]
}
return u[1],u[2]
```

### 3.1.1 Constraints

As we have previously seen, the original algorithm is a force algorithm that converges to the eigenvectors of the graph. It consists of three parts that have been needed to adapt to apply the following constraints:

1. **D-orthogonalization**. This constraint is imposed on each iteration so as to add more information.
2. **Multiplication for** $\frac{1}{2}(Id + D^{-1}M)$. So that it converges to the suitable eigenvector.
3. **Normalization**. Although it can be imposed at the end of the algorithm, it is imposed at each iteration so that the vectors do not explode.

The result of this algorithm to a slightly more complex graph that the one of image 5 without further constraints is:

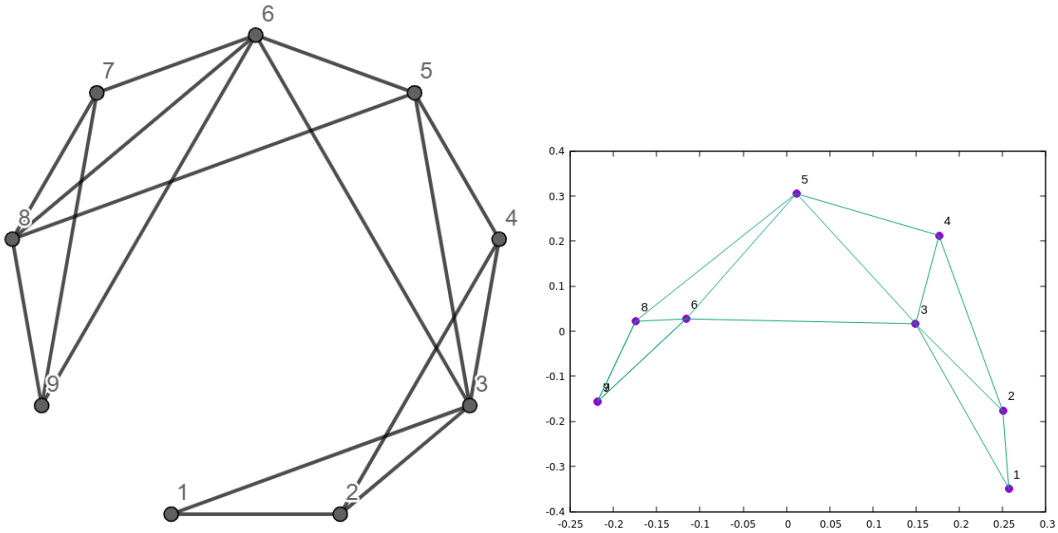


Figure 7: The optimized graph (right) and the initial one (left)

On what follows, we will revisit the constraints previously presented, and explain how each of the three parts have been adapted. Finally, an example of the graph 7 with the given constraints will be shown.

As we can see in figure 7, since vertices 7 and 9 have the same adjacencies, they are plotted at the same point. There are different solutions to this problem, from adding some constraint as the ones we will see to one of those points, to changing the adjacencies or adding extra repulsion to one of the points.

The most difficult part to adapt is the first one, that is, to D-orthogonalize while maintaining the constraints. Two approaches have been developed to solve the problem, giving different results depending on the constraints. The first one will be explained later on and is a variation of the Gram-Schmidt method. The second is to model the optimization problem consisting in D-orthogonalising and minimising the distance to the non-orthogonal point. For instance in the case without any constraint, we would have to solve the following optimizaton problem:

$$\underset{\bar{x},\bar{y}\in\mathbb{R}^n}{\text{minimize}} \quad \sum_{i=1}^{n} (\bar{x}-x)^2 + (\bar{y}-y)^2$$

$$\text{such that:} \quad x^T D 1_n = y^T D 1_n = x^T D y = 0$$

For any other constraint, you only need to add some other constraints to the problem. For instance, for the fixed points, you only need to not allow the movement of those anchored points, excluding them from the variables; for the alignment points, just force the corresponding components to be equal, and so on. The optimization problem is then solved by an NLP solver, giving a good result in a few seconds in each iteration, but slowing down all the process. In fact, in the prior optimization problem we are not minimizing the distance, but the square of it. The reason for this is that, although our solver, Couenne

can understand the absolute value, given its exotic behaviour, the runtime was excessive.

To conclude, even though this D-orthogonalization gives better results in all of the cases except the fixed points constraint, we only use it to produce the final images. For each constraint, the result of using each of the two D-orthogonalization methods will be shown, and the differences compared.

**Fixed points.**

1. **D-orthonormalization.** The algorithm uses Gram-Schmidt with the dot product defined as $< x, y >= x^T D y$, but since this method does not maintain fixed coordinates, we need a modified version of this. First, suppose we want to orthogonalize $x$ with respect to $y$, where $x = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}$, $y = \begin{pmatrix} y_1 \\ y_2 \end{pmatrix}$, and $x_1 \in \mathcal{R}^s$ are the coordinates that we do not want to modify. Then it is sufficient to apply the transformation:

$$\bar{y} = \begin{pmatrix} y_1 \\ y_2 - \frac{y^T D x}{x_2^T D x_2} x_2 \end{pmatrix}$$

Check that $\bar{y}$ is orthogonal to $x$:

$$\begin{aligned} \bar{y}^T D x &= \begin{pmatrix} y_1 & y_2 - \frac{y^T D x}{x_2^T D_2 x_2} x_2 \end{pmatrix} \begin{pmatrix} D_1 & 0 \\ 0 & D_2 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \\ &= y_1^T D_1 x_1 + y_2^T D_2 x_2 - \frac{y^T D x}{x_2^T D_2 x_2} x_2 D_2 y_2 \\ &= y^T D x - \frac{y^T D x}{x_2^T D_2 x_2} x_2 D_2 y_2 = 0 \end{aligned}$$

Consider now that we also want to orthogonalize also $z = \begin{pmatrix} z_1 \\ z_2 \end{pmatrix}$ with respect to $x$ and $\bar{y}$. The first guess we could make could be

$$\bar{z} = f(z) = \begin{pmatrix} z_1 \\ z_2 - \frac{z^T D x}{x_2^T D_x x_2} x_2 - \frac{z^T D y}{y_2 D_2 y_2} y_2 \end{pmatrix}$$

But this does not result in an orthogonal vector, since

$$\begin{aligned} \bar{z}^T D x &= -\frac{z^T D y}{y_2^T D_2 x_2} (y_2^T D_2 x_2) \\ \bar{z}^T D \bar{y} &= -\frac{z^T D x}{x_2 D_2 x_2} (x_2^T D_2 y_2) \end{aligned}$$

Even so, the recursion $z^{(k+1)} = f^2(z^{(k)})$ converges to $\bar{z}$, orthogonal vector to $\bar{y}$ and $x$.

$$z^{(k)^T} D x = \frac{(z^{(k)^T} D x)(y_2^T D x_2)^2}{(y_2^T D_2 y_2)(x_2^T D_2 x_2)} = \ldots = \frac{(z^T D y)(y_2^T D - 2 x_2)^{2k}}{(x_2^T D_2 x_2)^k (y_2^T D_2 y_2)^k}$$

Now, if we consider the dot product $< x_2, y_2 >= x_2^T D_2 y_2$, it is held that $(x_2^T D_2 y_2)^2 \leq < x_2, x_2 >< y_2, y_2 >= (x_2^T D_2 x_2)(y_2^T D_2 y_2)$ and so $x^{(k)^T} D x \to 0$. Using the same procedure, one can also check

that $x^{(k)^T} D\bar{y} \to 0$. In fact, this converge is really quick, usually taking two iterations at most.

The algorithm to D-orthogonalize vector $u1$ with respect to $u[0]$ and $u[1]$ while maintaining fixed the components of $u1_2$ and changing $u1_1$, is:

```
for(l=0;l<=j;l++)
    u1_1 = u1_1 - (u1_1^T D_1 u[l]_1)/(u[l]_1^T D_1 u[l]_1) u[l]_1

for(j=0;j<itermax and (|res1|>tolerance or |res2|>tolerance);j++){
    for(l=0;l<j;l++)
        u1_1 = u1_1 - (u1_1^T D_1 u[l]_1)/(u[l]_1^T D_1 u[l]_1) u[l]_1
    res1=u1^T D 1_N
    res2=u1^T D u1
}
```

2. **Multiplication for $(Id + D^{-1}M)$.** No change has been needed besides not taking into account the result of the operation for the fixed coordinates. After the multiplication we have empirically found out that it works much better if we also orthogonalize.

3. **Normalization:** No modification has been needed, since what is important is the relative position with respect to the others points, but not the absolute position.

The result of this method applied to the original graph but with the constraints that vertices 3 and 5 are adjacent to the north; 1 and 3 to the east; 2, 7 and 9 to the south, and 6 the the west, is:

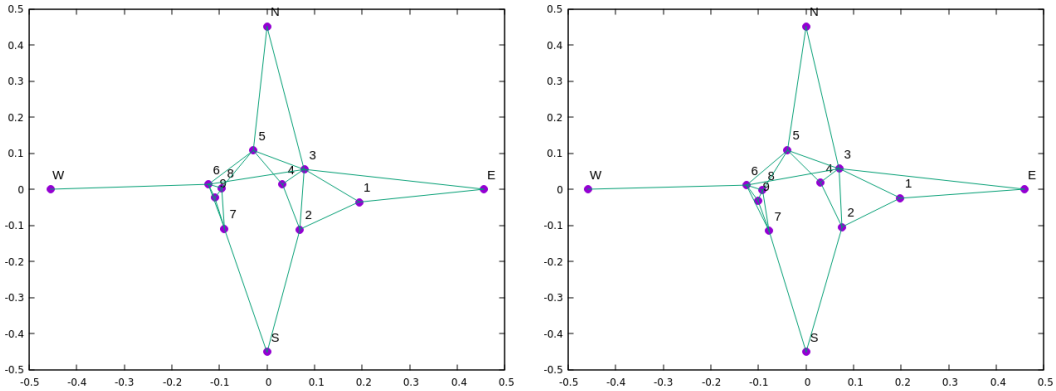


Figure 8: N, S, E and W are fixed points using the explained D-orthogonalization method (left) and the NLP model (right)

Comparing the result of this method and the best optimum found when introducing directly all the variables and constraint to the solver Couenne they are just a bit different. However, if you compute each of the axis independently and compare them with the corresponding optimization model to the solver, you find that in each direction we have a global optimum.

No difference can be seen between the two images. However, the small differences make that our way of D-orthogonalize produce slightly better results than using the NLP solver in each iteration to D-orthogonalize. This was expected, as our method is an adaptation to the one given in paper [8]. If more iterations were done in the NLP solver method, the same results would be found. This will not be the case of any of the others constraints, in which using a optimization solver will lead to better and highly different results.

**Vertical/horitzontal alignment.**

1. **D-orthogonalization:** The D-orthogonalization can move the points outside the line. Because of that, we D-orthogonalize considering the aligned points as fixed points.

2. **Multiplication for $\frac{1}{2}(Id + D^{-1}M)$:** After multiplying, we look for the more adequate regression line. To do so, as it is an horizontal or vertical line, we only need to find the middle point between the other coordinates. After the multiplication we need to D-orthogonalize again.

3. **Normalization:** Since when normalizing the constraint is maintained, no further change is needed.

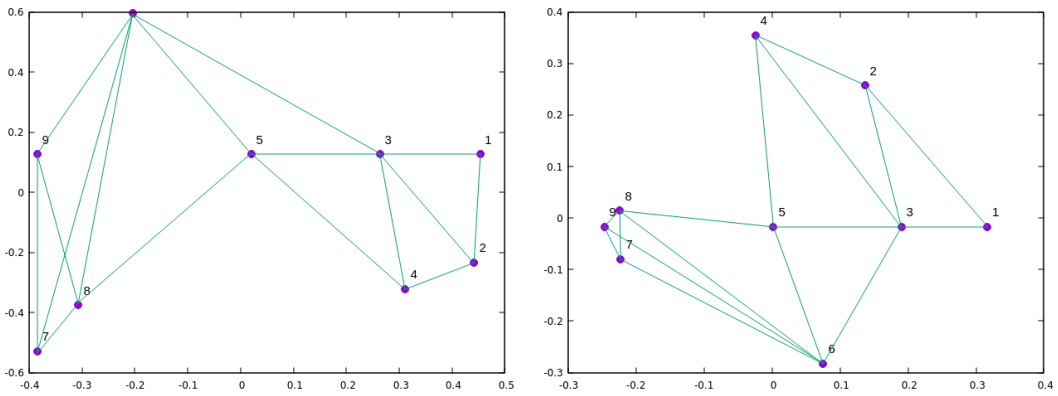


Figure 9: Vertices 1, 3, 5 and 9 are aligned at the same horizontal line. Using fixed points D-orthogonalization (left) and NLP solver (right)

We can see that the results are significantly different depending on which method we use, although applying an horizontal symmetry we can see that the distribution is not completely different. Also the result is better when using the NLP optimization method when D-orthogonalizing. The image of the global optimum found introducing the model to a solver is:

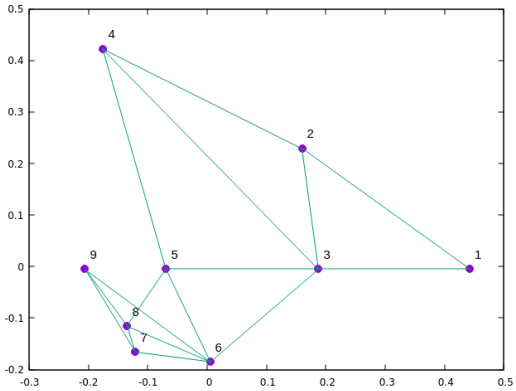


Figure 10: Global optimum with 1, 3, 5 and 9 aligned points.

**Clustering.**
The procedure to produce a cluster does not change any of the parts of the algorithm, but the graph to be introduced in it. The steps are:

1. Add a new vertex for each cluster, named $V_c$.
2. Add attractive forces between the vertices of the cluster and $V_c$.
3. Add repulsive forces between the vertices that are not in the cluster $c$ and $V_c$ and between attractors.
4. Apply the usual algorithm without further modifications.

An example of this could be to cluster the first four vertices and the five last ones. The result of the algorithm is:

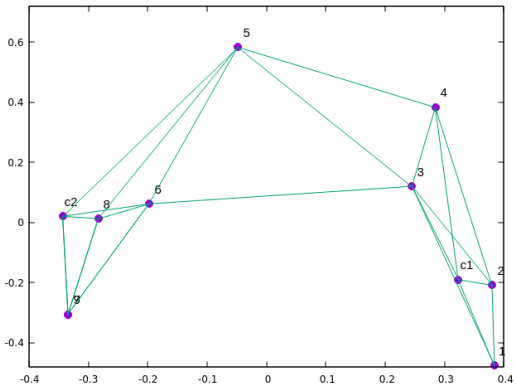


Figure 11: Cluster of the fourth first and fifth last vertices. c1 and c2 are the attractors of each of the clusters.

**Fixed subgraph**
The set of points in the subgraph is addressed as if it was a rigid body. One can apply transformations to it, the same transformation for all of the points.

1. **D-orthogonalize.** Since in general we can not guarantee that the relations between the vertices are the ones defined by the subgraph, the algorithm addresses these points as fixed points.

2. **Multiply by** $(1/2)\left(Id + D^{-1}A\right)$**.** The following proceeding is unidimensional so it has to be applied to each of the axes separately. We compute where each vertex should be placed, even the ones in the subgraph, but we do not move them. At the end, we compute which composition of transformations (homotheties, translations or rotations) makes the points before the multiplication be as close as possible to the points after the multiplication. Consider $x$ the vector with the initial coordinates and $y$ after the multiplication. We want to find $\hat{y}$ that minimizes the distance to $y$ while maintaining the relationships. The objective is to find $\hat{y} = ax + b$ such that

$$\min\left\{\sum_{i\in I}(\hat{y}-y)^2\right\} = \min_{a,b}\left\{\sum_{i\in I}(ax+b-y)^2\right\}$$

   Where $I$ is the set of points on the subgraph. The solution to this problem is obviously the same as the solution to the problem of finding the best fitting line, that is,

$$a = \frac{\sum_{i\in I}(x_i-\bar{x})(y_i-\bar{y})}{\sum_{i\in I}(x_i-\bar{x})^2}, \qquad b = \bar{y} - a\bar{x}$$

   Since we do the square of the remainder, we do not get the best option but just a good approximation. It is better to D-orthogonalize after multiplying so that it can converge better.

3. **Normalization.** No change to this part has been done, maintains the relation between vertices, is just to apply an homothety.

An example of this constraint could be, for instance, to impose that vertex 7 has to be the midpoint of vertex 5 and 6. That could be the case in which we want a room, for example, a bathroom to be exactly between two other rooms, like a pair of bedrooms. The result of the algorithm is:

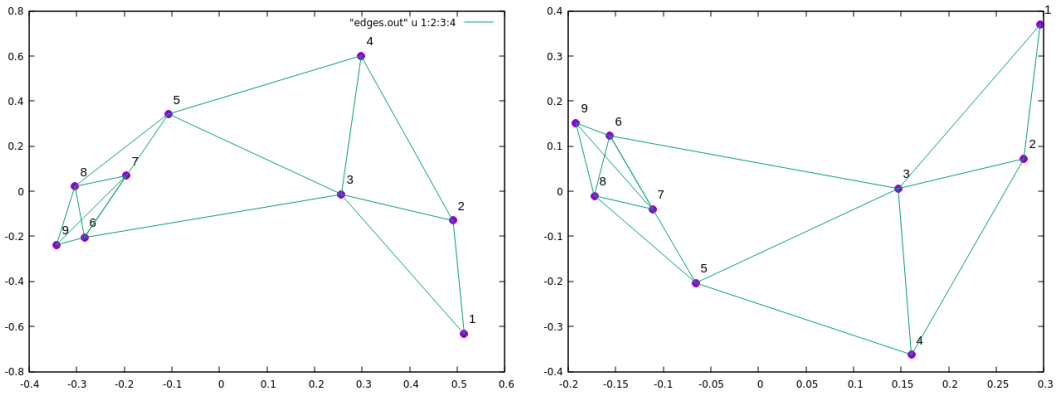


Figure 12: Vertex 7 is the midpoint of 5 and 6. D-orthogonalizing using our algorithm (left) and D-orthogonalizing optimization model (right)

The comparison with the global optimum obtained introducing the minimization problem to an NLP solver reveals that it is better to D-orthogonalize using an NLP solver. The global minimum is quite different from the others images. It is:

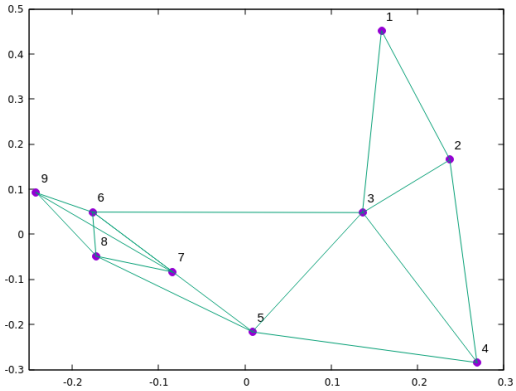


Figure 13: Global optimum with Couenne. Vertex 7 is the midpoint of 5 and 6.

**Constraint to a given convex region.**
Consider that we want to restrict one of the vertices to a given rectangle. The method is:

1. **D-orthogonalize.** Since the D-orthogonalization could move the points outside the rectangle, we treat them as a fixed point.

2. **Multiplication by** $(1/2)\left(Id + D^{-1}A\right)$**.** Compute where the point $x$ should go, call it $y$. Then by cases:

   - If $y$ is inside the rectangle or on the boundary, we keep $y$.
   - If $x$ is at the interior of the rectangle (not on the boundary) and $y$ outside it, we make the line between $x$ and $y$ and intersect with the boundary of the rectangle.
   - If $x$ is on the boundary of the rectangle and the new point outside it, we can move freely until the point on the boundary is closer to $y$.

   For stability reasons, it is better to D-orthogonalize after multiplying.

3. **Normalization.** Find the module of the vector, call it *mod*. Normalize the vector diving by *mod* and then apply the same homothecy to the points of the rectangle.

The case where we want to constraint the vertex 1 to the rectangle $[-0.107, 0.134] \times [-0.209, -0.157]$ gives the following graph:

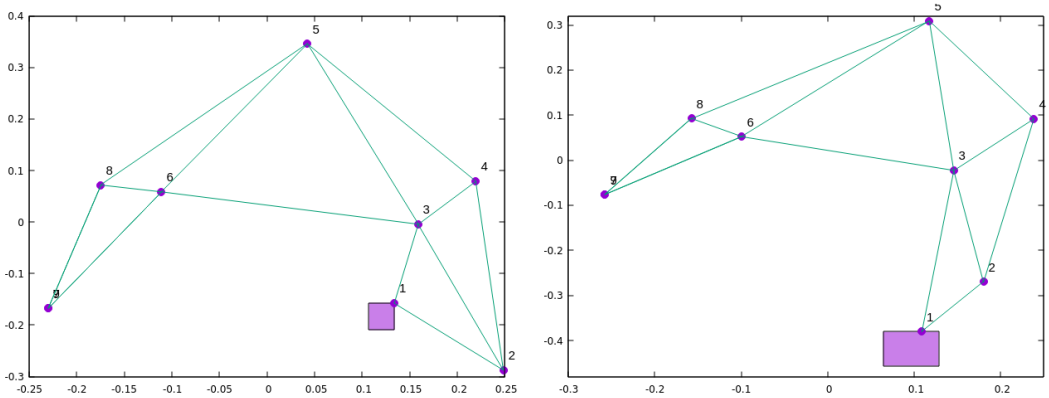

Figure 14: Vertex 1 is constrained in the lilac rectangle. Using D-orthogonalization with our method (left) and the optimization model method (right)

The output when introducing the complete model to an NLP solver reveals that, in this example, it is better to use our D-orthogonalization algorithm for fixed points. This could be because there is only one point to be anchored. The solution found by introducing the complete model to our NLP solver is:

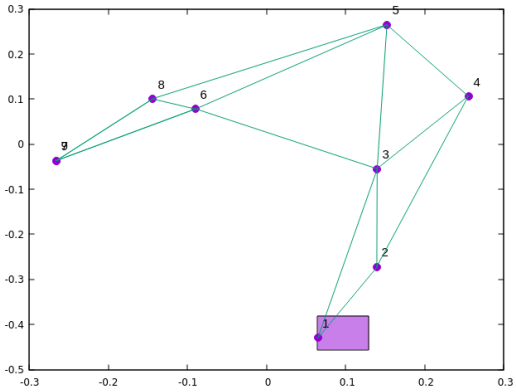

Figure 15: Vertex 1 is constrained in the lilac rectangle. Global optimum.

Finally, the following table summarize the loss percentage between the minimum obtained with our algorithm and the global minimum found by an optimization solver:

| | Fixed points orthogonality | Modeled orthogonality |
|---|---|---|
| Fixed points | 0.093% | 0.039% |
| Vert. aligment | 8.84% | 2.60% |
| Midpoint | 0.78% | 0.19% |
| Inside rectangle | 2.64% | 1.44% |

In all these constraints, when we use Gram-Schmidt to D-orthogonalize we could be searching not in all the space of feasible solutions, but only in a subspace of it. whereas when D-orthogonalizing with an NLP model we are searching in all the space. To conclude, we may say that to produce the final output it is better to D-orthogonalize modeling with an NLP model since, although the complexity increases, the

output found is better.

## 3.2 Definition of the rooms' walls

Now that we have the optimum location of the centers of the rooms, we should find the dimensions of the rooms that minimize a given objective function. This, in fact, is not done in this project and here we only intend to produce a good input to be optimized by another algorithm. One of the main problems that the algorithm could have, has to deal with intersecting rooms. This problem has been solved using different approaches. For instance, for some type of graphs you could construct its rectangular dual, as done in paper [6], another way of tackling the problem is reading it as a packing problem. However, our approach will be to model the problem as an optimization problem and solving it using a NLP solver, as Couenne. The models introduced to the solver are programmed using C.

Two different models will be studied trying to minimize the intersection. Which of them is the best will depend on the algorithm that will be used to find the global optimum and the constraints and our interests.

On the first model, we impose the rooms not to overlap and we find the minimum boundary box, whereas on the second model, we set a boundary box and move the rooms so that the overlay is minimized. Although the more realistic model is the second one, since most of the times the architect can not produce the apartment where he pleases, but has a fixed space for it; the first is also interesting and may be useful depending on the algorithm.

Another aspect that should be tackled at this point would be the connections between rooms and the use of corridors. There are different approaches to this problem but is a complex problem that will not be solved in this project

### 3.2.1 Finding the bounding box of minimum size

Suppose that you need the best apartment's layout given the conditions from scratch, without any prior boundaries. In this case, you may want to find the smallest bounding box that can enclose all the rectangles, as it is done in [3], where some of the ideas of the model have been taken. The "smallest bounding box" can be computed either minimizing its area or its perimeter. Some constraints have to be done so that we construct an architectural layout:

**Relative position:** Given an initial position of the rectangles, do not permit the relative position to change, that is, if a room is on the right of another one it has to be maintained this way.

The initial dimensions of the rooms is set by the user. The rectangles representing the rooms of the initial input should not intersect and should be large, since they will define the future relationships. Depending on this first input, the final result may vary, so has to be chosen carefully. In this project we propose to choose them as large squares that do not intersect any other object and without moving the centers of the rooms, that should be placed on the vertices found in the above chapter, although any other

initial input would be equally useful. For instance, for the example we are working with, we can use the following initial dimensions of the rooms:

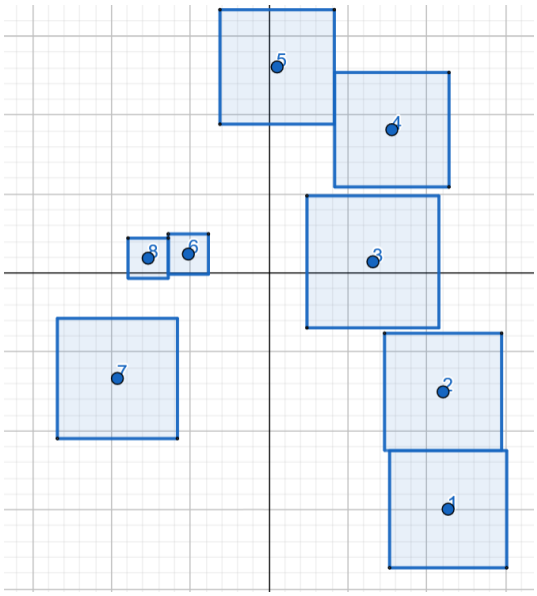


Figure 16: Initial position of rectangles

After this graph is done, we need to specify the relative position constraint, that is, which rooms are on the left of which or on the top of which others... But imposing all possible constraints would require too many conditions. For instance, vertex 3 is at the right of vertex 6, 7 and 8. Then we have to impose:

$$x_6 + \frac{1}{2}w_6 \leq x_3 - \frac{1}{2}w_3 \tag{3}$$

$$x_7 + \frac{1}{2}w_7 \leq x_3 - \frac{1}{2}w_3 \tag{4}$$

$$x_8 + \frac{1}{2}w_8 \leq x_3 - \frac{1}{2}w_3 \tag{5}$$

On the other hand, since rectangle 8 is on the left of rectangle 6, we will also have to include the condition

$$x_8 + \frac{1}{2}w_8 \leq x_6 - \frac{1}{2}w_6 \tag{6}$$

However, using the transitive rule, condition 5 can be eliminated. The following graph explains more clearly and without repetitions, the relative position conditions:

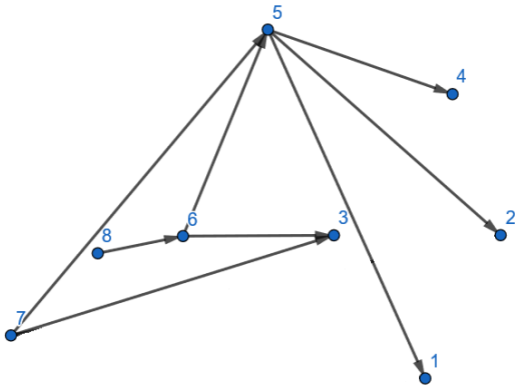


Figure 17: Horizontal relative positions graph

This graph explains the horizontal relative conditions, and in the same way the vertical ones can also be constructed. The constraints that need to be added then are just the maximal oriented paths of the graph, without repeating edges. The algorithm could just be to compare all the pairs of vertices and add a edge if needed, and then delete all the repeated paths, but we have also propose an algorithm based on geometric programming and the sweeping line algorithm with complexity $\mathcal{O}(n^2)$. For instance, to find the horizontal relative position graph could be:

1. Project the rectangles in the x-axis and order all the starting $s_i$ and ending $e_i$ points of the rectangles. Then, for each of the rectangles:
2. Start a sweeping line at $s_i$ and go to the left direction and search for the first endpoint, let it be $e_j$.
3. Add the edge $(j, i)$ to the horizontal relative position graph, since rectangle $j$ is at the left of $i$.
4. Continue the sweeping line through the left until you find $s_j$, adding the rectangle $k$ if $e_k$ is between $s_j$ and $e_j$.

**No intersection.** The walls of the rooms should not intersect. No further conditions need to be imposed as they are implicitly defined by the constraints of relative position.

**Aspect ratio:** Impose an aspect ratio for each of the rooms.

$$l_i w_i \leq h_i \leq u_i w_i$$

Where $l_i, w_i$ are chosen parameters that can be distinct for each of the rooms. One can use this constraint to impose rooms to be functional, that is, a bedroom with height 0.5m may not make sense, but a corridor with this proportions could be possible.

**Minimum area:** To satisfy our design needs, we can also impose a minimum area for each of the rooms:

$$w_i h_i \geq A_i$$

Finally we only need to find the objective function. As we have to minimize the area of the bounding box, we first need to compute the bounding box's length and wide. Consider the simple initial graph

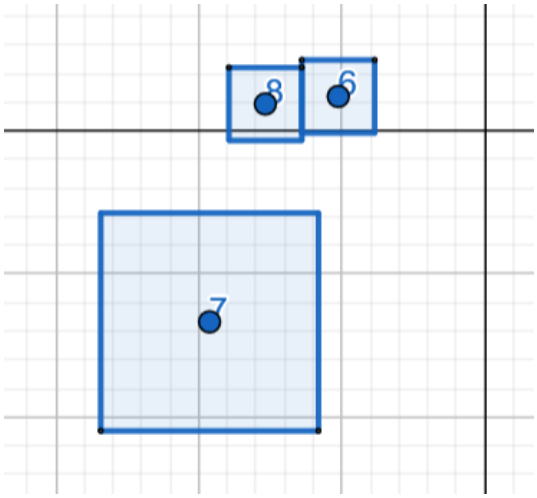


Figure 18: Caption

Since we have the following constraints:

- Room 8 has to be on the left of room 6.
- Room 7 has to be under rooms 6 and 8.

Then, we can compute the area of its bounding box as

$$\begin{aligned} W &= \max\{w_7, w_6 + w_8\} \\ H &= \max\{h_6 + h_7, h_8 + h_7\} \\ \text{Àrea} = WH &= \max\{w_7(h_6 + h_7), w_7(h_8 + h_7), (w_6 + w_8)(h_6 + h_7), (w_6 + w_8)(h_8 + h_7)\} \end{aligned}$$

Couenne does not understand the functions max or min, so we need to rewrite the Area making use of new constraints. This is done creating a new variable, say $Obj$, which has to be greater or equal to all the components of the maximum. In this small example it would be:

$$\begin{aligned} &\underset{w,h\in\mathbb{R}^3}{\text{minimize}} && Obj \\ &\text{such that:} && Obj \geq w_7(h_6 + h_7),\ Obj \geq w_7(h_8 + h_7) \\ & && Obj \geq (w_6 + w_8)(h_6 + h_7) \text{ and } Obj \geq (w_6 + w_8)(h_8 + h_7) \end{aligned}$$

Introducing the described requirements into a model for the NLP solver, we get the global optimum in a small amount of time and it is the following:

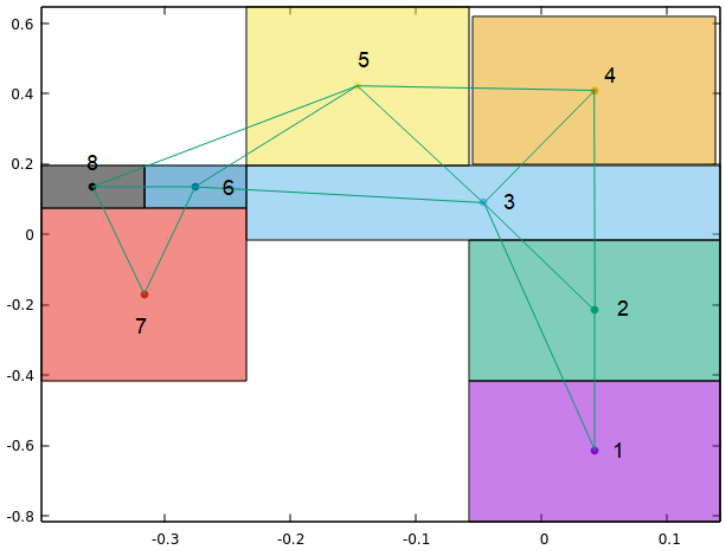


Figure 19: Solution to model of minimizing area of bounding box

Some of the rooms that we preferred to be in contact in the graph preferences, do not touch, for instance rectangles 3 and 7, or on the contrary (rooms 5 and 8), but this can be corrected by changing the initial rectangle graph. On the other hand, the centers of the rooms of the final output are near the ones computed as initial output. So to summarise, it may be a good model if you do not have a prior boundary rectangle.

### 3.2.2 Minimizing the intersection

Suppose now that we have a predefined boundary box, the space where we can construct the apartment, and we want to find an initial position with minimum intersection to introduce to a gradient based algorithm. Some of the constraints are the same as in the previous model, concretely the minimum area and the aspect ratio constraint. The non-intersection constraint has to be obviously eliminated. Finally, the relative position constraint has to be modified and a new constraint has to be added.

**Relative position:** As most of the rectangles do intersect, they are neither on the left nor right, and we will have to consider the relative positions between the centers of the rectangles. We do not need to write the $n(n-1)$ constraints, but only $2n$: for each point we just consider the relative position between it and the nearest point on the right and upper.

**Boundary box:** In this model we have a fixed rectangular boundary box and all rooms have to be inside it. If the boundary box is $[W^d, W^u] \times [H^d, H^u]$ then the constraint that has to be added for every $i$ is:

$$W^d \leq x_i - 0.5w_i, \qquad x_i + 0.5w_i \leq W^u$$
$$H^d \leq y_i - 0.5h_i, \qquad x_i + 0.5h_i \leq H^u$$

Then we need to find the objective formula to minimize, that is, the area of intersection. Since the possible combinations that can have two rectangles that intersect are:

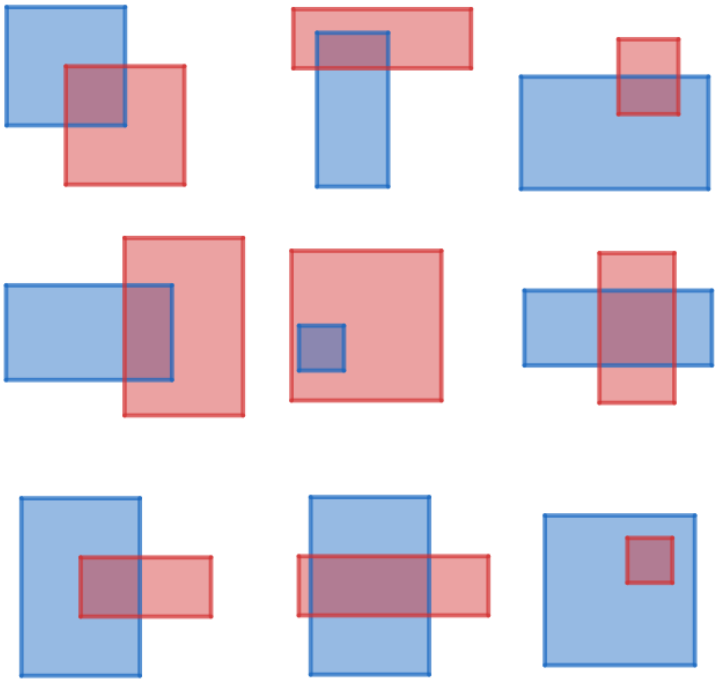

Figure 20: Possible types of intersections between a pair of rectangles

Let $i$ and $j$ be two intersecting rectangles and let $i$ be on the left of $j$. Then the intersection can be expressed as:

$$\min\{x_i - x_j + 0.5 * (w_i + w_j), w_i, w_j\} \min\{-|y_i - y_j| + 0.5 * (h_i + h_j), h_i, h_j\}$$

But this function of computing the area of intersection has two big problems. First, it produces a greater than zero output when there is no intersection between the rectangles and second, it can not be model with simple functions without the use of maximum or minimum functions. Despite this, it can be simplified in such a way that it corrects the two problems:

$$I_{ij} = \max\{0, (x_i - x_j + 0.5 * (w_i + w_j))\} \max\{0, (-|y_i - y_j| + 0.5 * (h_i + h_j))\}$$

Where the maximum is done to have an area of intersection 0 if rectangles do not intersect. Obviously, this does not compute the correct area of intersection, but it is a good enough approximation of it. This can be modeled as:

$$
\begin{aligned}
&I_{ij}^x \geq 0 \\
&I_{ij}^x \geq (x_i - x_j + 0.5 * (w_i + w_j)) \\
&I_{ij}^y \geq 0 \\
&I_{ij}^y \geq (-|y_i - y_j| + 0.5 * (h_i + h_j)) \\
&I_{ij} = I_{ij}^x I_{ij}^y
\end{aligned}
$$

Finally, the function to minimize is:

$$I = \sum_{i:x_i<x_j} I_{ij}$$

As an example of the output of the model, we have considered the same examples as before. The minimum areas of each room are set proportional to its degree in the graph. Finally, the height of the rectangle has been set to be at minimum a half and no more than three times its width. The global optimum of this optimization problem, computed with Couenne, is:

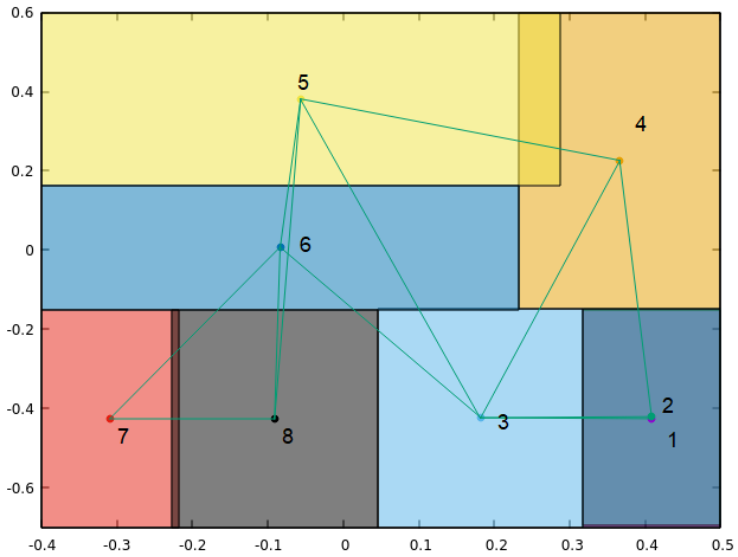


Figure 21: Solution to the no intersection model

This image is a good optimum but the layout is very different from the initial one and the properties may not be maintained. In order to correct this, we may add to the objective function a weighing of the square of the remainder, that is,

$$k \sum_{i=1}^{n} (x_i - \bar{x}_i)^2 + (y_i - \bar{y}_i)^2, \; k \geq 0$$

Where $\bar{x}, \bar{y}$ are the solution to the layout of the centers of the rooms found in chapter 3.1; and $k \geq 0$ is the weighing we want to give to the proximity of the solution to the initial one. In the following images the problem with the same parameters as before is solved using a different weighing of the proximity function:

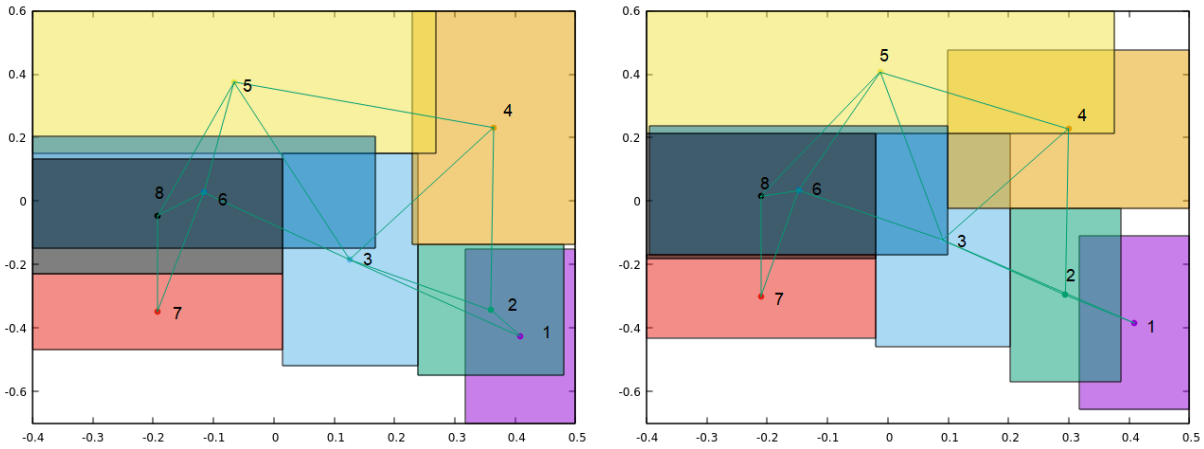


Figure 22: Solution to the no intersection model for $k = 0.5$ (left) and $k = 1$ (right)

As we can see, as we put more importance to the proximity to the initial layout graph, the rectangles layout loses the rectangular structure but gains the resemblance to the initial graph.

# 4. Example: A medium sized apartment

Until now we have only presented a simple example used to illustrate the ideas we wanted to explain, but without any touch with real floor plans. In this chapter, however, we will take ideas from a real floor plan and compare the output obtained with our algorithms with the reality. The programs to produce the figures and results of this section are written in C and are in the appendixes B,C and D.

The apartment choosen was a 4-bedrooms apartment. Its owners asked for the following properties of the apartment:

- The space that can occupy the apartment can have windows in any direction but in the south, although can only have terraces on the east or west.
- we want the flat to have two separate spaces, one public with the kitchen, a studio, a dinning room and a terrace; and one more private with the bedrooms and a private studio.
- We need 3 bedrooms.
- We need 2 bathrooms, one to be used both for the family and their guests, and one next to the main bedroom.
- We want the studio that is in the public space to be at the west space, so that it has better light at the afternoon. The other studio to be near the main bedroom.
- We would prefer the living room to be at the east, so taht it has better light in the morning.
- A terrace in each of the possible direction, (east and west).
- Finally, the main bedroom to be at the east.

This requirements can be mathematically modeled with the following constraints:

1. Add two fixed points. One on the right and one on the left. Add the following adjacencies:
   - East: The living room, one terrace, the main bedroom.
   - West: Studio, one terrace.
2. Add two clusters. One to cluster the private rooms and the other one the public ones.
3. Finally, restrict the main bedroom, another bedroom and both bathrooms to be horizontally aligned.

In total, then, we have to construct a 15-vertices graph. The meaning of each vertex and the approximate area that occupy its corresponding room in the reality are:

| Vertex | What | Area | Vertex | What | Area | Vertex | What | Area |
|---|---|---|---|---|---|---|---|---|
| 1 | Bedroom | 17 | 6 | Priv. Studio | 6 | 11 | Studio | 10 |
| 2 | Main Br | 21 | 7 | Terrace W. | 6.5 | E | East | |
| 3 | Priv. Wc | 6 | 8 | Kitchen | 13.5 | W | West | |
| 4 | Bathroom | 5 | 9 | Living room | 26 | C1 | cluster 1 | |
| 5 | Bedroom | 8 | 10 | Terrace E. | 9 | C2 | cluster 2 | |

After applying the algorithm described in section 3.1 with the previously described constraints, the graph obtained is:

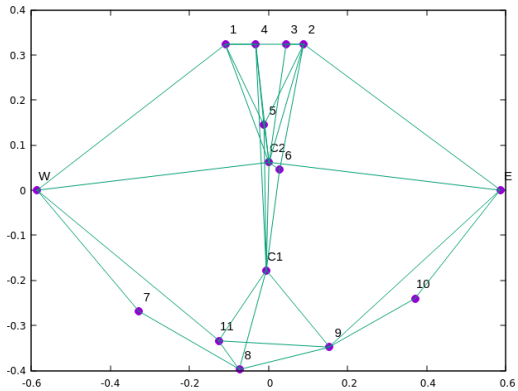


Figure 23: Output graph with the fixed points, W and E and the cluster attractos points, c1 and c2

As we can see, this graph is not very sparse, but follows all the conditions needed. Our algorithm output has a minimum near the global minimum. Instead of the value for the objective function, $x^T Lx + y^T Ly$ as explained in chapter 3.1, that we get, 0.9776 the global minimum is less than 0.9072. In fact, the global minimum could not be found by our solver due to the time complexity.

Although in this project we have not dealt with corridors, and they should be added a posteriori, in this example, and for visual purposes, we will set the second cluster to be the entrance hallway and the first to be a corridor.

Different solutions for the problem of minimizing the intersection with different degrees of influence on the distance between the centers of the initial graph and the actual centers of the rectangle are:

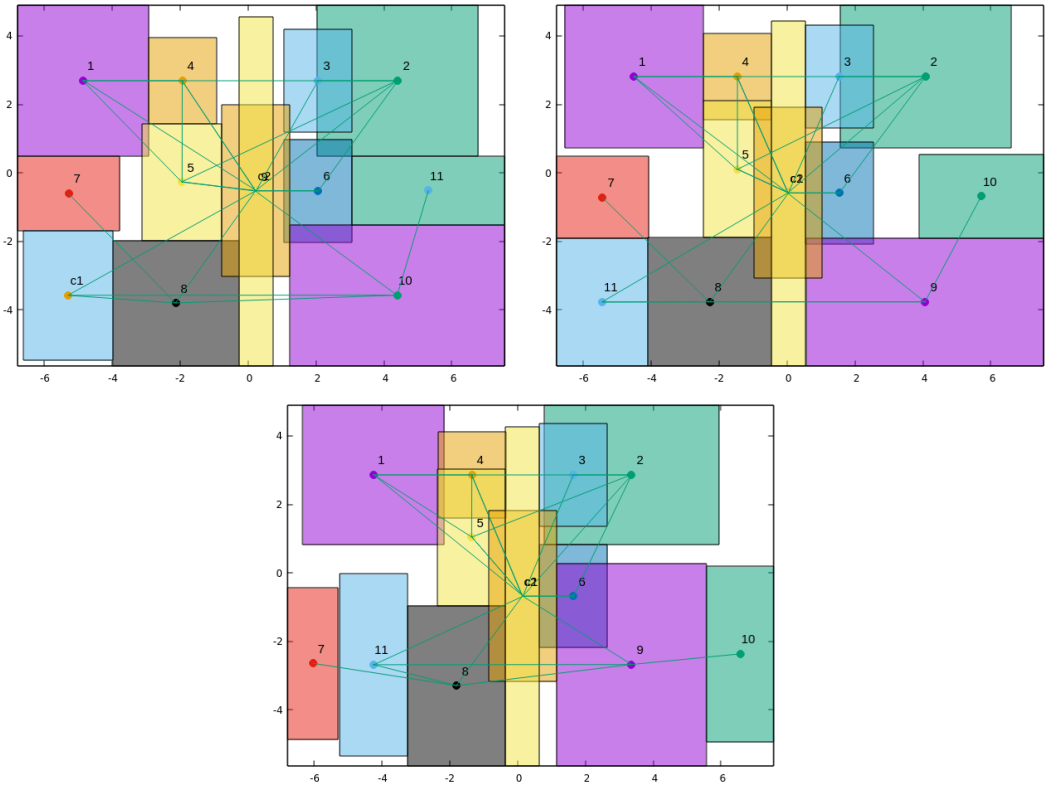


Figure 24: Minimum intersection. Degree of influence of 0 (left), 0.5 (right) and 1 (lower)

We could see that the approximated intersection areas was almost exact for all pairs of rectangles but for the case c1, c2. In this case the approximated value was double the currently one. The other model, which impose no intersection and minimizes the area of the bounding box, gives as a result:

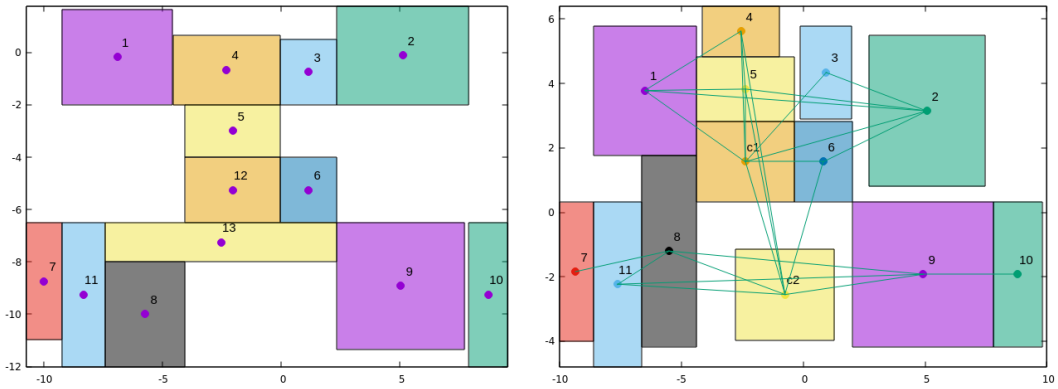


Figure 25: Minimum boundary box.

The left image is produced using the initial rectangles as explained in chapter 3.2.1, that is, squares as large as possible, whereas the right one uses a slightly different initial rectangles, the ones that seemed better to us after different attempts.

None of the produced floor plans is feasible, although seems to be a good start of another algorithm, and is more or less similar to the real floor plan:


Figure 26: Floor plan of a real apartment

# 5. Conclusion and prespective

In this project we have tried to give a good initial layout to solve the complex problem of floor planning. First, we have used spectral graph drawing in order to find the best location of the centers of the rooms given the design preferences. We have also adapted this algorithm to different requirements and constraints that could be encountered when planning an apartment. Although sometimes in this case we could not find the global minimum, in all the studied cases the difference was small enough. Two algorithms have been studied for the process of D-orthogonalization, since it is where the problems arises the most: if we orthogonalize using Gram-Schmidt, it seems that we are not looking at the entire set of feasible solutions, but on a subset of this, thus maybe not finding any minimum; whereas modelling the problem as an optimization one and solving it through some nonlinear solver, usually led to better results.
After we have found the best layout of the centers of the rooms, we have created a pair of models to preprocess the data before applying another algorithm — that has not been developed in this project — that would find the minimum. In this way, we still can maintain the good properties created but minimize the intersection or the boundary box, depending on our needs.

## 5.1 Future work

None of the results of our project is by itself a feasible floor plan, but it is one that, maintaining all our preferences, could be a good initial layout to another algorithm. This part should be further studied, comparing the results of our initial layout and other types of layouts, and weighing the aesthetics properties that have been created using the force algorithm.

Another part that should be deeply studied is the model of the corridors. In this work, we have tried two approaches. The first, is to include in the design preferences the corridors, and require that every room is adjacent to its assign corridor. The second, is to use clusters, if needed, to group rooms to the same corridor. Another third option, that is used in some papers, is to find the minimal tree that goes through all the rooms of the graph. In this project we have not seen which method is better to use, and the answer may vary from problem to problem.

## 5.2 Acquired knowledge

In this project I have been introduced to the problem of floor planning and also to graph drawing and spectral graph theory, which I did not know beforehand. I have also learned about NLP solvers and the modeling language AMPL, practising and learning about mathematical optimization. Nevertheless, the most challenging part was to think about the variations of the algorithms to better fit the new constraints and the examples, as well as solving some of the programming problems.

# References


[1] Scott A Arvin and Donald H House. Modeling architectural design objectives in physically based space planning. *Automation in Construction*, 11(2):213 – 225, 2002. ACADIA '99.

[2] Giuseppe Di Battista, Peter Eades, Roberto Tamassia, and Ioannis G. Tollis. *Graph Drawing: Algorithms for the Visualization of Graphs, chapter 10*. Prentice Hall PTR, USA, 1st edition, 1998.

[3] Stephen Boyd and Lieven Vandenberghe. *Convex Optimization, chapter 8*. Cambridge University Press, USA, 2004.

[4] Jose Duarte. Customizing mass housing : a discursive grammar for siza's malagueira houses. *European computer aided architectural design and education*, 08 2005.

[5] Kernighan B.W. Fourer R., Gay D.M. *AMPL: A Mathematical Programing Language*. Springer, 1989.

[6] Goos Kant and Xin He. Two algorithms for finding rectangular duals of planar graphs. 790, 07 1994.

[7] Michael Kaufmann and Dorothea Wagner. *Drawing graphs. Methods and models, chapter 4*, volume 2025. Springer, 01 2001.

[8] Y. Koren. Drawing graphs by eigenvectors: theory and practice. *Computers & Mathematics with Applications*, 49(11):1867 – 1888, 2005.

[9] Danny Lobos and Dirk Donath. The problem of space layout in architecture: A survey and reflections. *Arquitetura Revista*, 6:136–161, 12 2010.

[10] Wenqi Huang Mao Chen. A two-level search algorithm for 2d rectangular packing problem. *Computers & Industrial Engineering*, 53(1):123 – 136, 2007.

[11] Jeremy Michalek, R. Choudhary, and Panos Papalambros. Architectural layout design optimization. *Engineering Optimization*, 34:461–484, 09 2002.

[12] Krishnendra Shekhawat. Algorithm for constructing an optimally connected rectangular floor plan. *Frontiers of Architectural Research*, 3(3):324 – 330, 2014.

[13] Nitant Upasani, Krishnendra Shekhawat, and Garv Sachdeva. Automated generation of dimensioned rectangular floorplans. *Automation in Construction*, 113:103149, 2020.

[14] Machi Zawidzki and Jacek Szklarski. Multi-objective optimization of the floor plan of a single story family house considering position and orientation. *Advances in Engineering Software*, 141:102766, 2020.

# A. Proofs of propositions

**Lemma 3.1** *Consider $v_1, \dots, v_{p-1} \in \mathbb{R}^n$ and $X \in \mathbb{R}^{n \times p}$ a matrix whose columns are orthogonal. Then there exists a matrix $Y$ whose columns are also orthogonal and holds:*

1. *For all $2 \leq k \leq p$, the column $k$ of $Y$, $Y^k$ is orthogonal to $v_1, \dots, v_{k-1}$.*
2. *For all $n \times n$ matrix $A$, $tr\,(X^T AX) = tr\,(Y^T AY)$.*

*Proof.* Let $\bar{v}_1$ be the projection of $v_1$ into $< X_1, \dots, X_p >$. By cases:
If $\bar{v}_1 = 0$, then set $Y_1 = X_1$, and $\bar{Y}_2 = X_2, \dots, \bar{Y}_p = X^p$. Since $X$ is an orthogonal matrix, then $\bar{Y}_2, \dots, \bar{Y}_p$ are orthogonal to $Y_1$.
If $\bar{v}_1 \neq 0$, we rotate the vectors $X_1, \dots, X_p$ within $\mathrm{Rg}(X)$ such that, $Y_1 = \bar{v}_1/||\bar{v}_1||$, obtaining $Y_1, \bar{Y}_2, \dots, \bar{Y}_p$. Since rotations within all vectors do not change its orthogonality, $\bar{Y}_2, \dots, \bar{Y}_p$ are still orthogonal to $Y_1$. Recursively we find $Y_1, \dots, Y_p$ which satisfy the first requirement. On the other hand, since you apply rotations,there exists an $p \times p$, such that $Y = XR$, then,

$$Id = Y^T Y = R^T X^T XR = R^T R = Id$$

So,

$$\mathrm{tr}(Y^T AY) = \mathrm{tr}(R^T x^T AXR) = \mathrm{tr}(RR^T X^T AX) = \mathrm{tr}(X^T AX)$$

□

**Theorem 3.2** *Consider a symmetric positive definite $n \times n$ matrix $A$ and a diagonal positive definite matrix $B \in \mathbb{R}^{n \times n}$ and $p < n$. The solution of the optimization problem*

$$\begin{aligned} \underset{x_1,\dots,x_p \in \mathbb{R}^n}{\textit{minimize}} \quad & \frac{\sum_{i=1}^{n} x_i^T A x_i}{\sum_{i=1}^{n} x_i^T B x_i} \\ \textit{such that:} \quad & x_1^T B x_1 = \cdots = x_p^T B x_p \\ & x_i^T B x_j = 0 \quad \forall 1 \leq i, j \leq p \end{aligned}$$

*is the smallest generalized eigenvectors of (A,B).*

*Proof.* As the objective function is invariant under homothetic transformations on $x$, we can set its module, after finding the solution $x_1^T Bx_1 = \cdots = x_p^T Bx_p = 1$ and so the problem becomes, without loss of generality,

$$\begin{aligned} \underset{x_1,\dots,x_p \in \mathbb{R}^n}{\text{minimize}} \quad & \sum_{i=1}^{n} x_i^T A x_i \\ \text{such that:} \quad & x_i^T B x_j = \delta_{ij} \quad \forall 1 \leq i, j \leq p \end{aligned}$$

Now, as $B$ is a symmetric matrix, we can apply a Cholesky transformation and there exists $C$ such as $B = C^T C$. Define $y_i = Cx_i$, the problem can be rewritten as

$$\begin{aligned} \underset{y_1,\dots,y_p \in \mathbb{R}^n}{\text{minimize}} \quad & \sum_{i=1}^{n} y_i^T C^{-T} A C^T y_i \\ \text{such that:} \quad & y_i^T y_j = \delta_{ij} \quad \forall 1 \leq i,j \leq p \end{aligned}$$

Now consider $\lambda_1 \leq \cdots \leq \lambda_n$ the eigenvalues of $M = C^{-T}AC^{-1}$ and $v_1, \dots, v_n$ its eigenvectors. Let $X$ be the matrix of $y_1, \dots, y_p$ column-wise, a $n \times p$ matrix. Then the problem is

$$\begin{aligned} \underset{X \in \mathbb{R}^{n \times p}}{\text{minimize}} \quad & \text{tr}\left(X^T A X\right) \\ \text{such that:} \quad & X^T X = Id \end{aligned}$$

Be $w = (w_1, \dots, w_p)$ the minimum. As $v_1, \dots, v_n$ are an orthogonal base of $\mathbb{R}^n$, we can write $w_i = \sum_{j=1}^{n} \alpha_{ij} v_j$. For the lemma we have that for all $2 \leq i \leq p$, $w_i$ is orthogonal to $v_1, \dots, v_{i-1}$. Thus, we can take $\alpha_{ij}$ for $i < j$ and $w_i = \sum_{j=i+1}^{n} \alpha_{ij} v_j$.

Using the constraint $w_i^T w_i = 1$,

$$1 = w_i^T v_i = \left(\sum_{j=i+1}^{n} \alpha_{ij} v_j\right)^T \left(\sum_{l=i+1}^{n} \alpha_{il} v_l\right) = \sum_{j=i+1}^{n} \sum_{l=i+1}^{n} \alpha_{ij}\alpha_{il} v_i^T v_j = \sum_{j=i+1}^{n} (\alpha_{ij})^2$$

The last equality is due to the fact that $v_i^T v_j = \delta_{ij}$. Then,

$$\begin{aligned} w_i^T M w_i &= \left(\sum_{j=i+1}^{n} \alpha_{ji} v_j\right)^T M \left(\sum_{l=j+1}^{n} \alpha_{li} v_l\right) = \left(\sum_{j=i+1}^{n} \alpha_{ji} v_j\right)^T \left(\sum_{l=j+1}^{n} \alpha_{li} A v_l\right) \\ &= \left(\sum_{j=i+1}^{n} \alpha_{ji} v_j\right)^T \left(\sum_{l=j+1}^{n} \alpha_{li}\lambda_l v_l\right) = \sum_{j=i+1}^{n} \sum_{l=i+1}^{n} \alpha_{ji}\alpha_{li}\lambda_l v_j^T v_l \\ &= \sum_{j=i+1}^{n} (\alpha_{ji})^2 \lambda_l \geq \sum_{j=i+1}^{n} (\alpha ji)^2 \lambda_i = \lambda_i \end{aligned}$$

Then, $\text{tr}\,(X^T M X) \geq \sum_{i=1}^{n} \lambda_i$ but as $\sum_{i=1}^{n} v_i^T M v_i = \sum_{i=1}^{n} \lambda_i$, the minimum is the first $p$ eigenvectors of $M = C^{-T}AC^{-1}$. We have to invert the change and get the solution for $x$. The eigenvectors of $C^{-T}AC^{-1}$, that is, $y_i$ such as $C^{-T}AC^{-1}w_i = \lambda_i w_i$, so the $x_i$ which holds $Ax_i = \lambda_i B x_i$.

Finally, we can conclude, as we wanted to proof, that the solution corresponds to the $p$ first generalized eigenvectors corresponding to the $p$ smallest generalized eigenvalues of (A,B). □

**Proposition 3.3** *The generalized eigenvectors of* $(L, D)$ *are also the generalized eigenvectors of* $(A, D)$ *with reverse order.*

*Proof.* Consider a generalized eigenvalue $\lambda$ and eigenvector $x$ of $(L, D)$, then it holds $Lx = \lambda Dx$ but since $L = D - A$,

$$\begin{aligned}(D - A)x &= \lambda Dx \\ Ax &= (1 - \lambda)Dx\end{aligned}$$

As we already stated that the generalized eigenvalues of $(L, D)$ are in [-1,1], the eigenvectors of (A,D) are the same but in reverse order. □

# B. Program of the example of chapter 4

```
#include<stdio.h>
#include<stdlib.h>
#include<math.h>
#define N 16
#define P 3 /*N #points, P-1 dimension to draw*/

/*D-orthogonalignedze using Couenne*/
double Dorton(int alignedgned[N], int fix[N], double u[P-1][N], int D[N][N]);
/*Computes the modulus*/
double modul(double []);

/*Input: file input, with the adjacency matrix.
  Output: file output, with the coordinates of each of the vertices*/
/* To compute de top eigenvectors of D^-1 A*/
int main(void){

  int l, i, j, iter, k;
  double u1[P-1][N], u[P][N], mod, num, den;
  double tolerance = 1-1.e-17;
  double product[2];
  double D[N][N], w[N][N];
  char ent[] = "input", sor[] = "output";
  FILE *input, *output;
  /*ent (input) will be the adjacency matrix of the graph,
  sor (output) the output, the optimum location of the points*/
  int aligned[N] = {1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0}; /*Points to be alignedgned*/
  int C = 2; /*Number of clusters*/
  int clus[2][N] = {{1,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0},
    {0,0,0,1,0,0,1,1,1,1,1,0,0,0,0}}; /*Clusters*/
  int fix[N] = {0,0,0,0,0,0,0,0,0,0,0,0,1,1,0}; /*Fixed points*/
  double u2[P-1][N] =
    {{1,-1,1,-1,1,0,1,1,1,1,1,1,-1,-1},{1,1,1,1,1,1,1,-1,0,-1,1,1,0,0,-1,0}};

  /*Open files*/
  input = fopen(ent,"r");
  output = fopen(sor,"w");

  /*The first vector is inizialized to 1 (is the first eigenvector, already known)*/
  for (i=0;i<N;i++)
    u[0][i] = 1.;
  mod = modul(u[j]);
  for (i=0;i<N;i++)
    u[0][i] = u[0][i] / mod;

  /*w is the adjacency matrix, D de degrees matrix*/
  for (i = 0; i < (N-C); i++) {
    for (j = 0; j < (N-C); j++) {
      D[i][j] = 0;
      fscanf(input, "%le ", &w[i][j]);
    }
  }

  /*Add the cluster points*/
  for (i = 0; i < C; i++) {
```

```
53      for (j = 0, num = 0; j < (N-C); j++) {
54        w[N-C+i][j] = clus[i][j];
55        w[j][N-C+i] = clus[i][j];
56        num = num + clus[i][j];
57      }
58      for (j = N-C; j < N; j++) {
59        w[N-C+i][j] = 0.;
60        D[N-C+i][N-C+i] = num;
61        u2[0][N-C+i] = 0.;
62        u2[1][N-C+i] = 0.;
63      }
64    }
65
66    /*Find the degree matrix, D*/
67    for (i = 0; i < N; i++) {
68      for (j = 0, num = 0; j < N; j++)
69        num += w[i][j];
70      D[i][i] = num;
71    }
72
73    /*Normalize*/
74    for (i = 0; i < (P-1); i++) {
75      mod = modul(u2[i]);
76      for (j = 0; j < N; j++)
77        u2[i][j] = u2[i][j] / mod;
78    }
79
80    product[0] = 0.; product[1] = 0.;
81
82    for (iter = 1; iter < 1000 && (product[0] < tolerance || product[1] < tolerance);
        iter++) {
83
84      for (i = 0; i < N; i++) {
85        for (j = 0; j < (P-1); j++)
86          u1[j][i] = u2[j][i];
87      }
88
89      Dorton(aligned, fix, u1, D);
90
91      /*Multiply with 0.5(I+D-1A)*/
92      for (l = 0; l < (P-1); l++) {
93        for (i = 0; i < N; i++) {
94          if (fix[i] == 0) { /*Do not move fixed points*/
95            for (j = 0, num = 0; j < N; j++) {
96              if (w[i][j] != 0)
97                num = num + w[i][j]*u1[l][j];
98            }
99            u2[l][i] = 0.5*(u1[l][i] + num/D[i][i]);
100         }
101       }
102     }
103
104     /*Impose aligned points*/
105     for (i = 0, num = 0; i < (N-2);i++) {
106       if (aligned[i] == 1)
107         num += u2[1][i];
108     }
```

```
109     for (i = 0; i < (N-2); i++) {
110       if (aligned[i] == 1)
111         u2[1][i] = num / 3.;
112     }
113
114     Dorton(aligned, fix, u2, D);
115
116     /*Normalization*/
117     for (j = 0; j < (P-1); j++) {
118       mod = modul(u2[j]);
119         for (i = 0; i < N; i++)
120           u2[j][i] = u2[j][i] / mod;
121     }
122
123     /*Compute the dot product*/
124     product[0] = 0; product[1] = 0;
125     for (j = 0; j < (P-1); j++) {
126       for (i = 0; i < N; i++)
127         product[j] = product[j] + u1[j][i]*u2[j][i];
128     }
129
130     /*Save the results*/
131     for (j = 0; j < (P-1); j++) {
132       for (i = 0; i < N; i++)
133         u[j+1][i] = u2[j][i];
134     }
135   }
136
137
138   printf("Final amb %d iteracions\n",iter);
139
140   /*D-normalize*/
141   for (i = 0; i < 2; i++) {
142     for (j = 0, mod = 0; j < N; j++)
143       mod = mod + D[j][j]*u[i+1][j]*u[i+1][j];
144     mod = sqrt(mod);
145     for (j = 0; j < N; j++)
146       u[i+1][j] = u[i+1][j] / mod;
147   }
148
149   /*Print the results*/
150   for (i = 0; i < N; i++) {
151     for (j = 1; j < P; j++)
152       fprintf(output, "%le ", u[j][i]);
153     fprintf(output, "\n");
154   }
155
156   printf("productFinal=(%le,%le)\n", product[0], product[1]);
157
158   return 0;
159 }
160
161 /*Input: a vector of N coordinates
162   Output: Its modulus (no its D-modulus)*/
163 double modul(double v[N]) {
164   double mod = 0;
165   int i;
```

```
166
167   for (i = 0; i < N; i++)
168     mod = mod + v[i]*v[i];
169
170   return sqrt(mod);
171 }
172
173 /*Input: the boolean vectors of the aligned and fixed points, the coordinates of the
      points and the degree matrix
174   Output: u will be returned with the new D-orthogonal more nearer points*/
175 /*This function D-orthogonalize the points of u so that the distance between the
      introduced points and the orthogonal ones is minimum*/
176 void Dorton(int aligned[N], int fix[N], double u[P-1][N], int D[N][N]){
177
178   int i, k; /*k will be used to compute de constraints*/
179   double firstAligned;
180   char cou[]="ortonormal.mod", csor[]="proau.out";
181   FILE *orton, *cout;
182   /*cou (orton) the model to introduce to the solver
183    csor (cout) where AMPL will write the solution to the optimization problem*/
184
185   orton = fopen(cou, "w");
186
187   /*Defining the variables*/
188   for (i = 0; i < N; i++)
189     if (fix[i] == 0)
190       fprintf(orton, "var x%d:=%le;\nvar y%d:=%le;\n", i+1, u[0][i], i+1, u[1][i]);
191     else
192       fprintf(orton, "param x%d:=%le;\nparam y%d:=%le;\n", i+1, u[0][i], i+1, u[1][i
      ]);
193
194   /*Minimization objective*/
195   fprintf(orton, "minimize obj: 0");
196   for (i = 0; i < N; i++)
197     fprintf(orton, "+ (x%d - %le)^2 + (y%d - %le)^2", i+1, u[0][i], i+1, u[1][i]);
198
199   /*Constraints*/
200   fprintf(orton, ";\nsubject to c1: 0");
201   for (i = 0; i < N; i++)
202     fprintf(orton, "+%le*x%d", D[i][i], i+1);
203   fprintf(orton, "=0;\nsubject to c2: 0");
204   for (i = 0; i < N; i++)
205     fprintf(orton, "+%le*y%d", D[i][i], i+1);
206   fprintf(orton, "=0;\nsubject to c3: 0");
207   for (i = 0; i < N; i++)
208     fprintf(orton, "+%le*x%d*y%d", D[i][i], i+1, i+1);
209   fprintf(orton, "=0;");
210
211   /*Aligned points: We search for the first aligned point of the vector aligned and
      then equal the other found with it*/
212   for (i = 0, k = 4, firstAligned= -1; i < N; i++) {
213     if (aligned[i] == 1) {
214       if (firstAligned == -1)
215         firstAligned = i;
216     } else {
217       fprintf(orton, "subject to c%d: y%d=y%d;\n", k, (int)(num+1), i+1);
218       k++;
```

```
219       }
220     }
221
222     /*Pass to Couenne*/
223     fclose(orton);
224     system("ampl <couenC.run");
225
226     /*Save the values*/
227     cout = fopen(csor,"r");
228     for (i = 0; i < N; i++)
229       fscanf(cout, "%le %le", &u[0][i], &u[1][i]);
230     fclose(cout);
231   }
```

# C. Program to produce the no-intersection model of chapter 4

```c
#include<stdlib.h>
#include<math.h>
#include<stdio.h>
#define N 13
#define P 3

/*input: a file (points.out) with the points obtained in the prior algorithm (graph)
  output: a file (modelI.mod) to introduce to Ampl and solve using couenne*/
/*This program writes the .mod file to solve the minimizing intersection model*/
int main(void){

  int i, j, k, fx[N][N], fy[N][N], I, J;
  double U[N][2], l[N], u[N], tolerance=1.e-5;
  FILE *input, *output;
  char ent[] = "points.out", sor[]="modelI.mod";
  int horizontalOrder[N] = {7, 11, 1, 8, 4, 5, 12, 13, 6, 3, 2, 9, 10};
  int verticalOrder[N] = {8, 9, 11, 7, 10, 12, 6, 13, 5, 1, 2, 3, 4};
  double A[N] = {17, 21, 6, 5, 8, 6, 6.5, 13.5, 26, 9, 10, 10, 7.5}; /*Minimum area*/
  int aligned[N]={1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0}

  /*Open the files*/
  input = fopen(ent, "r");
  output = fopen(sor, "w");

  /*As defined in the model, l*width <= heigh <= u*width*/
  for(i = 0; i < N; i++) {
    l[i] = 0.5;
    u[i] = 3.;
  }

  /*Defining variables*/
  for (i = 0; i < N; i++) {
    /*U is the points defined by prior algorithm, we multiply them by the values that
     make the points to have the same modulum as in the real layout*/
    fscanf(input, "%le %le", &U[i][0], &U[i][1]);
    U[i][0] = U[i][0] * 18.4840235; U[i][1] = U[i][1] * 9.83831153;
    fprintf(output, "var x%d:=%le;\nvar y%d:=%le;\n", i+1, U[i][0], i+1, U[i][1]);
    fprintf(output, "var h%d:=0.1, >=1.;\nvar w%d:=0.1, >=1.;\n", i+1, i+1);
  }

  /*Definition of the parameters*/
  for (i = 0; i < N; i++)
    fprintf(output, "param A%d=%le;\nparam l%d=%le;\nparam u%d=%le;\n", i+1, A[i], i
    +1, l[i], i+1, u[i]);

  /*fx(i,j) = 1 if j on the right of i*/
  /*fy(i,j) = 1 if j above i*/
  for (i = 0; i < N; i++) {
    for (j = 0; j < N; j++) {
      if ( fabs(U[i][0] - U[j][0]) < tolerance )
        fx[i][j] = 0;
```

```
50        else {
51          if(U[i][0] < U[j][0])
52            fx[i][j] = 1;
53          else
54            fx[i][j] = -1;
55        }
56        if ( fabs(U[i][1] - U[j][1]) < tolerance )
57          fy[i][j] = 0;
58        else {
59          if(U[i][1] < U[j][1])
60            fy[i][j] = 1;
61          else
62            fy[i][j] = -1;
63        }
64      }
65    }
66
67    /*As explained we need the auxiliar Sxij and Syij. Sij = Sxij*Syij and Sij is an
        approximation of the intersection between i and j*/
68    for (i = 0; i < N; i++) {
69      for (j = 0; j < N; j++) {
70        if (fx[i][j] == 1)
71          fprintf(output, "var S%do%d, >=0;\nvar Sx%do%d, >=0;\nvar Sy%do%d, >=0;\n", i
      +1, j+1, i+1, j+1, i+1, j+1);
72      }
73    }
74
75    /*Objective function: The sum of the intersections, Sij*/
76    fprintf(output, "minimize obj: ");
77    for (i = 0; i < N; i++) {
78      for (j = 0; j < N; j++) {
79        if (fx[i][j] == 1)
80          fprintf(output, "S%do%d + ", i+1, j+1);
81       }
82    }
83    for (i = 0; i < N; i++) {
84      if (Aligned[i] == 1)
85      /*Aligned points can share the same x and y variable*/
86        fprintf(output, "1*((x%d - %le)^2 + (y%d - %le)^2) + ", i+1, U[i][0], 1, U[i
      ][1]);
87      else
88        fprintf(output, "1*((x%d - %le)^2 + (y%d - %le)^2) + ", i+1, U[i][0], i+1, U[i
      ][1]);
89    }
90    fprintf(output, "0;\n");
91
92    /*Define the constraint to define the objective function*/
93    for (i = 0, k = 1; i < N; i++) {
94      for (j = 0; j < N; j++) {
95        if (fx[i][j] == 1) {
96          fprintf(output, "subject to c%d: Sx%do%d>=(x%d-x%d+0.5*w%d+0.5*w%d);\n", k, i
      +1, j+1, i+1, j+1, i+1, j+1);
97          /*I,J is 0 (the first aligned point) if the points are aligned. It's value if
       not*/
98          if (aligned[i] == 1) I=0;
99          else I=i;
100         if (aligned[j] == 1) J=0;
```

```
101           else J=j;
102           fprintf(output, "subject to c%d: Sy%do%d>=(%d)*(y%d-y%d)+0.5*(h%d+h%d);\n", k
      +1, i+1, j+1, fy[i][j], I+1, J+1, i+1, j+1);
103           fprintf(output, "subject to c%d: S%do%d=Sx%do%d*Sy%do%d;\n", k+2, i+1, j+1, i
      +1, j+1, i+1, j+1);
104           k = k + 3;
105         }
106       }
107     }
108
109     /*such that*/
110     /*Can no go out from its boundary box*/
111     fprintf(output, "subject to c%d: x%d+0.5*w%d<=7.56212223;\nsubject to c%d: x%d-0.5*
        w%d>=-6.783;\nsubject to c%d: y%d+0.5*h%d<=4.902;\nsubject to c%d: y%d-0.5*h%d
        >=-5.624;\n", k, horizontalOrder[N-1], H[N-1], k+1, horizontalOrder[0],
        horizontalOrder[0], k+2, V[N-1], V[N-1], k+3, V[0], V[0]);
112     k = k + 4;
113     fprintf(output, "subject to c%d: y%d+0.5*h%d<=4.902;\nsubject to c%d: y%d+0.5*h%d
        <=4.902;\nsubject to c%d: y%d+0.5*h%d<=4.902;\n", k, 1, 1, k+1, 1, 2, k+2, 1, 3);
114     k = k + 3;
115     fprintf(output, "subject to c%d: y%d+0.5*h%d<=4.902;\nsubject to c%d: y%d+0.5*h%d
        <=4.902;\n", k, 12, 12, k+1, 13, 13);
116     k = k + 2;
117     fprintf(output, "subject to c%d: y%d+0.5*h%d>=-5.624;\nsubject to c%d: y%d+0.5*h%d
        >=-5.624;\nsubject to c%d: y%d+0.5*h%d=-5.624;\nsubject to c%d: y%d+0.5*h%d
        >=-5.624;\n", k, 11, 11, k+1, 12, 12, k+2, 13, 13, k+3, 9, 9);
118     k = k + 4;
119
120     /*Maintain the order of the vertices*/
121     for (i = 0; i < (N-1); i++) {
122       fprintf(output, "subject to c%d: x%d<=x%d+0.005;\n", k, horizontalOrder[i],
        horizontalOrder[i+1]);
123       k = k + 1;
124       if (verticalOrder[i]!=1 && verticalOrder[i]!=2 && verticalOrder[i]!=3 &&
        verticalOrder[i]!=4){
125         fprintf(output, "subject to c%d: y%d<=y%d+0.005;\n", k, verticalOrder[i],
        verticalOrder[i+1]);
126         k = k+1;
127       }
128     }
129
130     /*Minimum area*/
131     for (i = 0; i < N; i++, k++)
132       fprintf(output, "subject to c%d: h%d*w%d>=A%d;\n", k, i+1, i+1, i+1);
133
134     /*Aspect ratio*/
135     for (i = 0; i < N-2; i++, k++)
136       fprintf(output, "subject to c%d: h%d<=w%d*u%d;\n", k, i+1, i+1, i+1);
137
138     return 0;
139 }
```

# D. Program to produce the minimum boundary box model of chapter 4

```
#include<stdlib.h>
#include<math.h>
#include<stdio.h>
#define N 13
#define P 3

/*input: a file (points.out) with the points obtained in the prior algorithm (graph)
  output: a file (modelBB.mod) to introduce to Ampl and solve using couenne*/
/*This program writes the .mod file to solve the minimizing boundary box model*/
int main(void){

  int i, j, k;
  double U[N][2], l[N], u[N];
  double A[N] = {17, 21, 6, 5, 8, 6, 6.5, 13.5, 26, 9, 10, 10, 7.5};
  FILE *input, *output;
  char ent[] = "points.out", sor[] = "modelBB.mod";
  /* squareSide is the initial side of the squares defined by the user.
     horizontal/verticalOrder is the pairs of order without repetition
     firstLastVert/Horitz are all the combinations of abovemost, lowestmost pairs and
    rightmost leftmost pairs*/
  double squareSide[N] = {1.4225288, 0.8133418, 0.8133418, 1.4140854, 0.8200969,
    0.50186972, 3.756937, 0.955928, 2.9424982, 4.012473, 0.955928, 0.50186972,
    2.9424982};
  int horizontalOrder[25][2] = { {1,4}, {1,5}, {1,12}, {2,10}, {3,2}, {3,9}, {4,3},
    {4,6}, {5,6}, {5,3}, {6,2}, {6,9}, {7,1}, {7,11}, {8,12}, {8,5}, {8,9}, {9,10},
    {11,12}, {11,13}, {11,4}, {11,8}, {12,3}, {12,6}, {13,9} };
  int verticalOrder[17][2] = { {1,5}, {2,5}, {3,5}, {4,5}, {5,12}, {5,6}, {6,7},
    {6,13}, {6,10}, {6,9}, {6,11}, {12,7}, {12,13}, {12,9}, {12,10}, {12,11}, {13,8}
    };
  int firstLastVert[20][2] = { {1,7}, {1,8}, {1,9}, {1,10}, {1,11}, {2,7}, {2,8},
    {2,9}, {2,10}, {2,11}, {3,7}, {3,8}, {3,9}, {3,10}, {3,11}, {4,7}, {4,8}, {4,9},
    {4,10}, {4,11} };
  int firstLastHoritz[1][2] = { {7,10} };

  /*Open the files*/
  input = fopen(ent, "r");
  sortida = fopen(sor, "w");

  /*As defined in the model, l*width <= heigh <= u*width*/
  for (i = 0; i < N; i++) {
    l[i] = 0.5;
    u[i] = 3.;
  }

  /*Defining variables*/
  for (i = 0; i < N; i++) {
    /*U is the points defined by prior algorithm, we multiply them by the values that
     make the points to have the same modulum as in the real layout*/
    fscanf(input, "%le %le", &U[i][0], &U[i][1]);
    U[i][0] = U[i][0] * 18.4840235; U[i][1] = U[i][1] * 9.83831153;
    fprintf(output, "var x%d:=%le, <=10, >=-10;\nvar y%d:=%le, <= 10, >= -10;\n", i
```

```
      +1, U[i][0], i+1, U[i][1]);
42     fprintf(output, "var h%d:=%le, >=1.5;\nvar w%d:=%le, >=1.5;\n", i+1, squareSide[i
      ], i+1, squareSide[i]);
43   }
44
45   /*Definition of the parameters*/
46   for (i = 0; i < N; i++)
47     fprintf(output, "param A%d=%le;\nparam l%d=%le;\nparam u%d=%le;\n", i+1, A[i], i
      +1, l[i], i+1, u[i]);
48
49   /*Objective function*/
50   fprintf(output, "var Ob;\n");
51   fprintf(output, "minimize obj: Ob;\n");
52
53   /*Constraints for the objective function (minimizing the area using the maximum
      length and width with the ordering)*/
54   for (i = 0, k = 1; i < 1; i++) {
55     for (j = 0; j < 20; j++, k++)
56       fprintf(output, "subject to c%d: Ob>=(x%d+w%d/2-x%d+w%d/2)*(y%d+h%d/2-y%d+h%d
      /2);\n", k, firstLastHoritz[i][1], firstLastHoritz[i][1], firstLastHoritz[i][0],
      firstLastHoritz[i][0], firstLastVert[j][0], firstLastVert[j][0], firstLastVert[j
      ][1], firstLastVert[j][1]);
57   }
58
59   /*Minimum area*/
60   for (i = 0; i < N; i++, k++)
61     fprintf(output, "subject to c%d: h%d*w%d>=A%d;\n", k, i+1, i+1, i+1);
62
63   /*Apect ratio*/
64   for(i = 0; i < N; i++, k = k+2)
65     fprintf(output, "subject to c%d: l%d*w%d<=h%d;\nsubject to c%d: h%d<=w%d*u%d;\n",
       k, i+1, i+1, i+1, k+1, i+1, i+1, i+1);
66
67   /*No intersection (first horizontal then vertical)*/
68   for (i = 0; i < 25; i++, k++)
69     fprintf(output, "subject to c%d: x%d+0.5*w%d<=x%d-0.5*w%d;\n", k, horizontalOrder
      [i][0], horizontalOrder[i][0], horizontalOrder[i][1], horizontalOrder[i][1]);
70   for (i = 0; i < 17; i++, k++)
71     fprintf(output, "subject to c%d: y%d-0.5*h%d>=y%d+0.5*h%d;\n", k, verticalOrder[i
      ][0], verticalOrder[i][0], verticalOrder[i][1], verticalOrder[i][1]);
72
73   return 0;
74 }
```